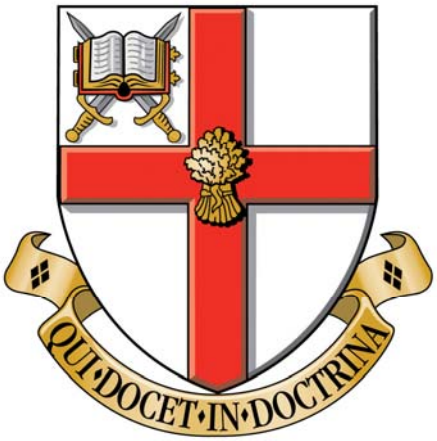


University of Chester

Tensor Decomposition and its Applications

Thesis submitted in accordance with requirements of the

University of Chester for the degree Master of Science

by Daniel Tock

September 2010

Acknowledgments

I would like to thank my Programme Leader Dr Jason Roberts whose encouragement, guidance and support throughout my thesis helped me to develop a wide understanding of the subject.

I also would like to thank my family and friends for their morale support and encouragement throughout the past year.

Abstract

In this Thesis I review classical vector - tensor analysis, building up to the necessary techniques required to decompose a tensor into a tensor train and to reconstruct it back into the original tensor with minimal error. The tensor train decomposition decomposes a tensor of dimensionality d into a train of d third order tensors, whose sizes are dependent upon the rank and chosen error bound. I will be reviewing the required operations of matricization, tensor - matrix, vector and tensor multiplication to be able to compute this decomposition.

I then move onto analysing the tensor train decomposition by applying it to different types of tensor, of differing dimensionality with a variety of accuracy bounds to investigate their influence on the time taken to complete the decomposition and the final absolute error.

Finally I explore a method to compute a d-dimensional integration from the tensor train, which will allow larger tensors to be integrated with the memory required dramatically reduced after the tensor is decomposed. I will be applying this technique to two tensors with different ranks and compare the efficiency and accuracy of integrating directly from the tensor to that of the tensor train decomposition.

Contents

1 Introduction

In today's society vector analysis, or more precisely, tensor analysis is one of the most important mathematical techniques used in the physical sciences and the modelling of real life situations. The concept of vectors were first introduced by the Egyptians and Babylons, but the first 2-dimensional and 3-dimensional vectoral systems weren't published until 1799 by Wessel [6]; hence the idea of a tensor (or a multidimensional array), is a fairly new mathematical concept.

Tensors or Multidimensional arrays are encountered in many real life applications, although due to the unmanageable amount of memory required we are unable to perform anything but basic operations upon them, with which the number of operations required to perform grow exponentially with the dimensionality d [12]. In order to reduce the required number of operations it is possible to express a tensor as the product of smaller tensors, hence reducing the memory required.

There are many forms of decomposition, the most popular of which is the canonical decomposition or parallel factors model, CANDECOMPPARAFAC decomposition [5]. The problem with this model is that it often suffers degeneracy (diverging components), and only has efficient algorithms for 3-dimensional data. For higher dimensional arrays the Tucker decomposition [5] is used which is effectively an extension of the CANDECOMPPARAFAC decomposition. This involves decomposing a d-dimensional tensor into a tensor of dimensionality $d-1$ and d 2-dimensional tensors (matrices). Again the issue of memory arises as for decomposing a tensor of large d, we still acquire a tensor of $d-1$ dimensions, which still requires a large amount of memory.

In an attempt to remove the issue of exponential memory requirements, I will be looking at the recently suggested Tensor-Train decomposition by E. Tyrtyshnikov and I. Oseledets [11], [12]. It can be computed via standard decompositions such as the singular value decomposition (SVD) and the QR decomposition and only produces 3-dimensional tensors, which are easily manageable to perform operations on.

The Tensor train decomposition algorithm can be considered as an adap-

tive interpolation algorithm for a multivariate function given on a tensor grid [12], and so we are able to apply it to high-dimensional integration. I will be comparing the accuracy and efficiency of using the original tensor to compute the integration against the tensor train.

2 Review of classical Tensor Analysis

A tensor is a multidimensional array. More formally, an N-way or Nth-order tensor is an element of the tensor product of N vector spaces, each of which has its own coordinate system. This notion of tensors is not to be confused with tensors in physics and engineering (such as stress tensors), which are generally referred to as tensor fields in mathematics [9].

2.1 Vectors and Scalars

Definition 2.1. *A vector is a quantity having both magnitude and direction [13].*

Displacement, Velocity, Acceleration and Force are all examples of vectors.

Graphically a vector is represented by a line OP, which defines the direction. The Magnitude of the vector is the length of the line. The starting point of the line, O, is called the origin, or initial point of the vector. The head of the line, P, is called the terminal point or terminus.

Analytically a vector can be described as $\mathbf{A}$ or $\vec{A}$, with magnitudes $\mid \mathbf{A} \mid$, A or $\mid \vec{A} \mid$. I will be using The bold faced notation throughout this paper, Hence the vector OP will be denoted as $\mathbf{OP}$ with magnitude $\mid \mathbf{OP} \mid$.

Definition 2.2. *A scalar is a quantity having only magnitude and no direction [13].*

Mass, Volume, Speed, energy, length, time and temperature are all examples of scalars. Scalars are defined by letters, and their operations follow the same rules as in elementary algebra.

2.1.1 Vector Algebra

With the following Definitions, we are able to extend the operations of addition, subtraction and multiplication to vector algebra.

Definition 2.3. *Two Vectors* $\mathbf{A}$ *and* $\mathbf{B}$ *are equal if they have the same magnitude and direction, regardless of the position of their initial points. As seen in figure 1.*

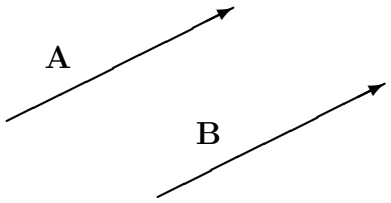


Figure 1: Equal vectors

Definition 2.4. *A vector having direction opposite to* $\mathbf{A}$ *but having the same magnitude is denoted by* $-\mathbf{A}$*. As seen in figure 2.*

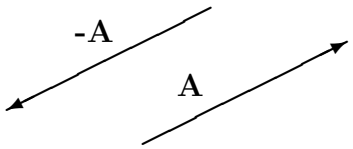


Figure 2: Opposite vectors

Definition 2.5. *The sum of two vectors* $\mathbf{A}$ *and* $\mathbf{B}$ *results in the vector* $\mathbf{C}$*, which is formed by placing the terminal point of* $\mathbf{A}$ *on the initial point of* $\mathbf{B}$*, and then joining the initial point of* $\mathbf{A}$ *and the terminal point of* $\mathbf{B}$*. As seen in figure 3.*

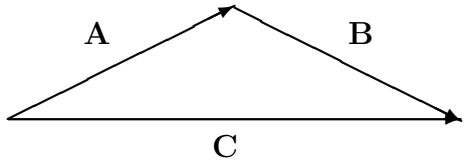


Figure 3: Sum of two vectors

Definition 2.6. *The difference of the two vectors* $\boldsymbol{A}$ *and* $\boldsymbol{B}$, $\boldsymbol{A}$-$\boldsymbol{B}$ *is equivalent to* $\boldsymbol{A}$+(-$\boldsymbol{B}$).

If $\mathbf{A} = \mathbf{B}$, then $\mathbf{A}$-$\mathbf{B}$ yields $\mathbf{0}$, the null or zero vector. $\mathbf{0}$ has zero magnitude and no specific direction. A vector which is not null is defined as a proper vector. All vectors are assumed to be proper unless otherwise stated [13].

Definition 2.7. *The product of a vector* $\boldsymbol{A}$ *by a scalar* m *is a vector* $m\boldsymbol{A}$ *with magnitude* $\mid m \mid \times \mid \boldsymbol{A} \mid$, *and direction the same as or opposite to that of* $\boldsymbol{A}$, *according to* m *being positive or negative respectively. If* $m = 0$, $m\boldsymbol{A}$ *is the null vector.*

2.1.2 Laws of Vector Algebra

If $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ are vectors and m and n are scalars then:

1. $\mathbf{A} + \mathbf{B} = \mathbf{B} + \mathbf{A}$ Addition is Commutative,
2. $\mathbf{A} + (\mathbf{B} + \mathbf{C}) = (\mathbf{A} + \mathbf{B}) + \mathbf{C}$ Addition is Associative,
3. $m\mathbf{A} = \mathbf{A}m$ Multiplication is Commutative,
4. $m(n\mathbf{A}) = (mn)\mathbf{A}$ Multiplication is Associative,
5. $(m + n)\mathbf{A} = m\mathbf{A} + n\mathbf{A}$ Distributive Law,
6. $m(\mathbf{A} + \mathbf{B}) = m\mathbf{A} + m\mathbf{B}$ Distributive Law.

Definition 2.8. ***A Unit Vector*** *has unit magnitude, i.e. a magnitude of 1. Let* $\boldsymbol{A}$ *be a proper vector, then* $\frac{\boldsymbol{A}}{|\boldsymbol{A}|}$ *is a unit vector with the same direction as* $\boldsymbol{A}$.

2.1.3 Cartesian Coordinate system

Any Vector in a 3-dimensional space can be described in terms of the **i**, **j** and **k** unit vectors, which relate to the x, y and z axes respectively.

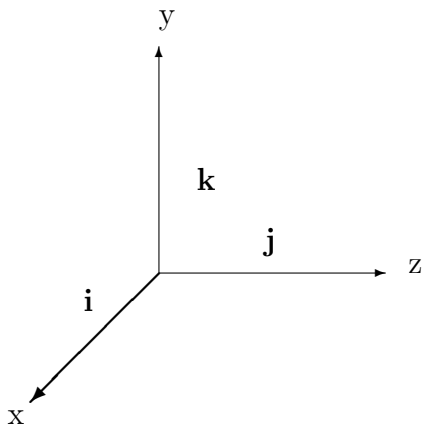


Figure 4: Cartesian Coordinate system

2.1.4 Contravariant and Covariant Vectors

If N quantities $A^1, A^2, ..., A^N$ in a coordinate system $(x_1, x_2, ..., x_N)$ are related to N other quantities $\bar{A}^1, \bar{A}^2, ..., \bar{A}^N$ in another coordinate system $(\bar{x}_1, \bar{x}_2, ..., \bar{x}_N)$ by the transformation equations:

$$\bar{A}^p = \sum_{q=1}^{N} \frac{\partial \bar{x}_p}{\partial x_q} A^q \qquad p = 1, 2, ..., N \tag{1}$$

Then they are called components of a contravariant vector.

Similarly, if N quantities $A^1, A^2, ..., A^N$ in a coordinate system $(x_1, x_2, ..., x_N)$ are related to N other quantities $\bar{A}^1, \bar{A}^2, ..., \bar{A}^N$ in another coordinate system $(\bar{x}_1, \bar{x}_2, ..., \bar{x}_N)$ by the transformation equations:

$$\bar{A}_p = \sum_{q=1}^{N} \frac{\partial x_q}{\partial \bar{x}_p} A_q \qquad p = 1, 2, ..., N \tag{2}$$

then they are called components of a covariant vector.

2.2 Tensor Analysis

Tensors are a further extension of ideas we already use when defining quantities such as scalars and vectors.

The order of a tensor is the dimensionality of the array that is required to represent it. A scalar, is a 0-dimensional array, therefore, can be represented by a tensor of order 0. A vector, is 1-dimensional array, and therefore can be represented by a 1st order tensor. A square matrix, is a 2 dimensional array, which represents a 2nd order tensor. In general a n-dimensional array is required to represent a nth order tensor.

2.2.1 The Einstein Summation Convention

This is a compact method of notation which omits the summation sign corresponding to any index which occurs twice in a single term. A repeated index in any term therefore implies a summation over that index [7]. We reserve the lowercase letters i, j, k, for indices ranging over the values 1, 2, 3 and shall not indicate their range explicitly.

For example, the equation $a_1x^1 + a_2x^2 + ... + a_Nx^N$ can be written as $\sum_{j=1}^{N} a_j x^j$. The Einstein summation convention shortens this yet further to $a_j x^j$, where $j = 1, 2, ..., N$.

2.2.2 The Kronecker Delta Symbol

The 2-dimensional Kronecker Delta Symbol, written as δ_{jk}, is defined by:

$$\delta_{jk} = \begin{cases} 0 & \text{if } k \neq j \\ 1 & \text{if } k = j \end{cases}$$

Properties of the Kronecker Delta Symbol from [2]:

- $\delta_{ij} = \delta_{ji}$
- $\delta^i_j = \delta^j_i = \delta^{ji} = \delta_{ji}$
- $\delta^i_i = 3$
- $\delta^j_i A_j = A_i$
- $\delta^{ij} A_i B_j = A^i B_i = \mathbf{A} \cdot \mathbf{B}$

2.2.3 The Levi-Civita Symbol

The 3-dimensional Levi-Civita symbol ϵ_{ijk} is defined as follows:

$$\epsilon_{ijk} = \begin{cases} +1 & \text{when } i,j,k \text{ is an even permutation of 1, 2, 3} \\ -1 & \text{when } i,j,k \text{ is an odd permutation of 1, 2, 3} \\ 0 & \text{when any two indices are identical} \end{cases}$$

The order i, j, k is an even/odd permutation of 1, 2, 3 if an even/odd number of transpositions is required to bring i, j, k into the order 1, 2, 3.

2.2.4 Contravariant, Covariant and Mixed Tensors

If N^2 quantities A^{qs} in a coordinate system $(x_1, x_2, ..., x_N)$ are related to N^2 other quantities $\bar{A}^{pr}$ in another coordinate system $(\bar{x}_1, \bar{x}_2, ..., \bar{x}_N)$ by the transformation equations:

$$\bar{A}^{pr} = \frac{\partial \bar{x}_p}{\partial x_q} \frac{\partial \bar{x}_r}{\partial x_s} A^{qs} \tag{3}$$

Then by the Einstein summation convention, they are called contravariant components of a tensor of rank 2.

Similarly, The N^2 quantities A_{qs} are called covariant components of a tensor of rank 2 if:

$$\bar{A}_{pr} = \frac{\partial x_q}{\partial \bar{x}_p} \frac{\partial x_s}{\partial \bar{x}_r} A_{qs} \tag{4}$$

We can also extend this idea to mixed tensors of higher ranks. For example, A^{qst}_{kl} are the components of a mixed tensor of rank 5. Contravariant of order 3 and covariant of order 2 if they transform to the relations:

$$\bar{A}_{ij}^{prm} = \frac{\partial \bar{x}_p}{\partial x_q}\frac{\partial \bar{x}_r}{\partial x_s}\frac{\partial \bar{x}_m}{\partial x_t}\frac{\partial x_k}{\partial \bar{x}_i}\frac{\partial x_l}{\partial \bar{x}_j}A_{kl}^{qst} \tag{5}$$

In any tensor equation an index can only appear once for a single index, or twice for a repeated index. i.e. A_{ii}^{i} is impossible. If an index is repeated, it may only be repeated once, i.e. A_{ii}^{ii} is impossible. A single index can be either be covariant or contravariant in the whole equation. It cannot be covariant in one term and contravariant in another [2].

2.2.5 Raising and Lowering Indices

The metric tensor can be used to raise and lower indices of a tensor.

Definition 2.9. ***Metric Tensor.*** *In Cartesian coordinates the square of the distance between two arbitrarily close points in space, one with coordinates x^i and the other $x^i + dx^i$, is:*

$$(ds)^2 = (dx)^2 + (dy)^2 + (dz)^2 = (dx^i)^2 \tag{6}$$

which can be written as:

$$(ds)^2 = \delta_{ij}dx^i dx^j \tag{7}$$

Hence the distance between these two point determines a tensor of rank 2, known as the metric tensor δ_{ij}. Thus, the covariant components of the metric tensor in Cartesian coordinates are given by Kronecker delta symbol [13].

For example, if A^i are contravariant components of a vector then its covariant components are $A_i = \delta_{ij}A^j$. Likewise $A^i = \delta^{ij}A_j$. This operation, called raising and lowering indices can be applied to any tensor.

2.2.6 Matricization

Matricization, also known as unfolding or flattening, is the process of reordering the elements of an N-dimensional array into a matrix [9]. For instance a $4 \times 3 \times 2$ matrix can be written as a 4×6 matrix, or a 8×3 matrix, and so on. For example consider the tensor $A \in \Re^{4\times 3\times 2}$ seen in figure 5.

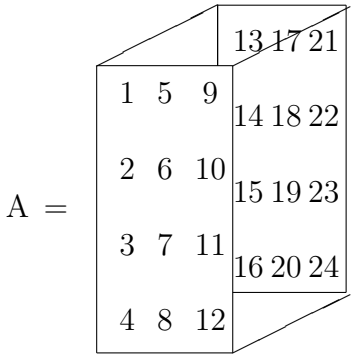


Figure 5: 3rd order, $4 \times 3 \times 2$ tensor,

For ease, I will display 3rd order tensors, as matrices, where the divisions represent the layers from the front face of the matrix to the back. Hence, A becomes:

$$A = \left(\begin{array}{ccc|ccc} 1 & 5 & 9 & 13 & 17 & 21 \\ 2 & 6 & 10 & 14 & 18 & 22 \\ 3 & 7 & 11 & 15 & 19 & 23 \\ 4 & 8 & 12 & 16 & 20 & 24 \end{array}\right)$$

There are three mode-n unfoldings of a 3rd order tensor, using A above as an example, we get:

$$A_{(1)} = \begin{pmatrix} 1 & 5 & 9 & 13 & 17 & 21 \\ 2 & 6 & 10 & 14 & 18 & 22 \\ 3 & 7 & 11 & 15 & 19 & 23 \\ 4 & 8 & 12 & 16 & 20 & 24 \end{pmatrix}$$

$$A_{(2)} = \begin{pmatrix} 1 & 2 & 3 & 4 & 13 & 14 & 15 & 16 \\ 5 & 6 & 7 & 8 & 17 & 18 & 19 & 20 \\ 9 & 10 & 11 & 12 & 21 & 22 & 23 & 24 \end{pmatrix}$$

$$A_{(3)} = \begin{pmatrix} 1 & 2 & 3 & 4 & ... & 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 & ... & 21 & 22 & 23 & 24 \end{pmatrix}$$

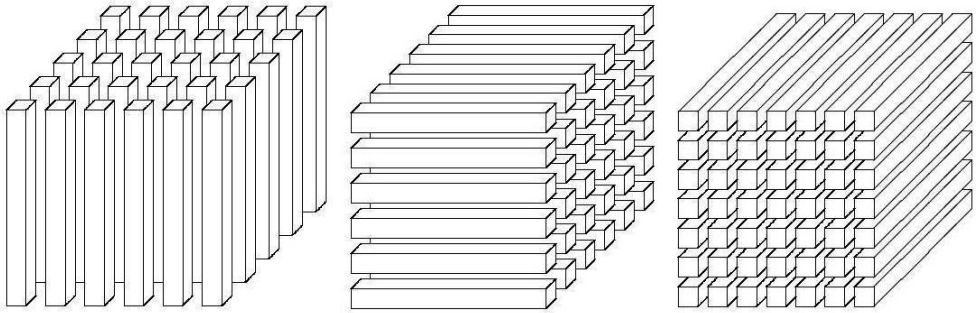
Different papers sometimes use different ordering of the columns for the mode-n unfolding. In general, the specific permutation of columns is not

important so long as it is consistent across related calculations.

It is also possible to vectorize a tensor. Once again, ordering the elements is not important as long as it is consistent.

$$Vec(A) = \begin{pmatrix} 1 \\ 2 \\ . \\ . \\ . \\ 24 \end{pmatrix}$$

Another way to imagine matricization is lining up the tensor fibres from a particular dimension as done by Kolda and Bader [8, 9].



(a) $A_{(1)}$ (b) $A_{(2)}$ (c) $A_{(3)}$

Figure 6: Fibres of a 3rd order tensor

Where the column fibres of a 3rd order tensor comprise the columns of $A_{(1)}$, as shown in figure 6a. The row fibres comprise the columns of $A_{(2)}$, as shown in figure 6b. And the tube fibres comprise the columns of $A_{(3)}$, as shown in figure 6c,

2.3 Tensor Multiplication

Tensors can be multiplied together, though obviously the notation and symbols for this are much more complex than for matrices.

2.3.1 Tensor-Matrix Multiplication

The n-mode product of a tensor $A \in \Re^{x_1 \times x_2 \times \ldots \times x_N}$ with a matrix $U \in \Re^{J \times x_n}$ is denoted by $A \times_n U$ and is of size $x_1 \times x_2 \times \ldots \times x_{n-1} \times J \times x_{n+1} \times \ldots \times x_N$.

Element wise we have:

$$(A \times_n U)_{i_1,\ldots,i_{n-1},j,i_{n+1},\ldots,i_N} = \sum_{i_n=1}^{x_n} a_{i_1,i_2,\ldots,i_N} u_{ji_n} \tag{8}$$

It is important to note that:

$$B = A \times_n U \tag{9}$$

relates to tensors, therefore after matricization, we use the following:

$$B_{(n)} = U A_{(n)}. \tag{10}$$

As an example, consider the tensor A defined above in figure 5 and let

$$U = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}.$$

Then the product of $A \in \Re^{4 \times 3 \times 2}$ with $U \in \Re^{2 \times 3}$ gives $B = A \times_2 U \in \Re^{4 \times 2 \times 2}$:

$$B_{(2)} = \begin{pmatrix} 61 & 70 & 79 & 88 & 169 & 178 & 187 & 196 \\ 76 & 88 & 100 & 112 & 220 & 232 & 244 & 256 \end{pmatrix}.$$

This is clearly not a $4 \times 2 \times 2$ tensor; to obtain the tensor we must reverse matricize B. It is of the form mode-2, due to the mode-2 multiplication that took place. Therefore we know that the columns of B comprise the row fibres of the tensor. Hence

$$B = \left[\begin{array}{cc|cc} 61 & 76 & 169 & 220 \\ 70 & 88 & 178 & 232 \\ 79 & 100 & 187 & 244 \\ 88 & 112 & 196 & 256 \end{array}\right].$$

There are a few important identities to note about n-mode multiplication:

- **Identity 1:** $A \times_n (UV) = A \times_n U \times_n V$,
- **Identity 2:** $A \times_m U \times_n V = A \times_n V \times_m U$ for $m \neq n$,
- **Identity 3:** Let U be a $p \times q$ matrix with full column rank. If $B = A \times_n U$ then $A = B \times_n Z$ where Z is the $q \times p$ left inverse of U.

For Example if U has orthonormal columns, then $B = A \times_n U \Rightarrow A = B \times_n U^T$.

2.3.2 Tensor-Vector Multiplication

The n-mode product of a tensor $A \in \Re^{x_1 \times x_2 \times \ldots \times x_N}$ with a vector $P \in \Re^{x_n}$ is again denoted by $A \times_n P$, or sometimes as $A \bar{\times}_n P$. The result is of order $N-1$ and is of size $x_1 \times x_2 \times \ldots \times x_{n-1} \times x_{n+1} \times \ldots \times x_N$.

Element wise we have:

$$(A \times_n P)_{i_1,\ldots,i_{n-1},i_{n+1},\ldots,i_N} = \sum_{i_n=1}^{x_n} a_{i_1,i_2,\ldots,i_N} P_{i_n} \tag{11}$$

The general idea is to compute the inner product of each mode-n fibre with the vector. Hence the equation:

$$C = A \times_n P \tag{12}$$

Which relates to the tensors, requires altering after Matricization. as we are computing the inner product we use the following equation:

$$C_{(n)} = P^T A_{(n)} \tag{13}$$

As an example, again consider the tensor A defined above in figure 5 and let

$$P = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}.$$

Then the product of $A \in \Re^{4 \times 3 \times 2}$ with $P \in \Re^4$ gives $C = A \times_1 P \in \Re^{3 \times 2}$:

$$C_{(1)} = \begin{pmatrix} 30 & 70 & 110 & 150 & 190 & 230 \end{pmatrix}$$

Again we must reverse matricize C to achieve the correct solution. It is of the form mode-1, therefore we know that the columns of C comprise the column fibres of the tensor. Hence

$$C = \left(\begin{array}{ccc|ccc} 30 & 70 & 110 & 150 & 190 & 230 \end{array}\right),$$

which is equivalent to

$$C = \begin{bmatrix} 30 & 70 & 110 \\ 150 & 190 & 230 \end{bmatrix}.$$

When it comes to mode-n vector multiplication, the following rule applies, where A is a tensor, and $\mathbf{b}$ and $\mathbf{c}$ are vectors:

$$A \times_n \mathbf{b} \times_m \mathbf{c} = (A \times_n \mathbf{b}) \times_{m-1} \mathbf{c} = (A \times_m \mathbf{c}) \times_n \mathbf{b} \quad \text{for} \quad m > n. \tag{14}$$

2.3.3 Tensor-Tensor Multiplication

The last category of tensor multiplication to consider is the product of two tensors. There are three general types of multiplication to consider for tensor-tensor multiplication: outer product, contracted product and inner product.

Tensor $\times$ Tensor Outer Product The outer product of two tensors is often just referred to as the tensor product. If we first consider it in terms of vectors, the outer product of two vectors $\mathbf{b} \in \Re^3$ and $\mathbf{c} \in \Re^5$ is defined by $\mathbf{d} = \mathbf{b} \otimes \mathbf{c} \in \Re^{3\times 5}$.

$$\mathbf{b} \otimes \mathbf{c} = \mathbf{b}\mathbf{c}^T$$

Element wise we get:

$$(\mathbf{b} \otimes \mathbf{c})ij = \mathbf{b}_i \mathbf{c}_j$$

For example, if we let

$$\mathbf{b} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \text{ and } \mathbf{c} = \begin{bmatrix} 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{bmatrix}$$

then their outer product is

$$\mathbf{b} \otimes \mathbf{c} = \mathbf{b}\mathbf{c}^T = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \begin{bmatrix} 4 & 5 & 6 & 7 & 8 \end{bmatrix} = \begin{bmatrix} 4 & 5 & 6 & 7 & 8 \\ 8 & 10 & 12 & 14 & 16 \\ 12 & 15 & 18 & 21 & 24 \end{bmatrix}.$$

Similarly, the outer product of two tensors $U \in \Re^{u_1 \times \ldots \times u_N}$ and $V \in \Re^{v_1 \times \ldots \times v_M}$ gives $A = U \otimes V \in \Re^{u_1 \times \ldots \times u_N \times v_1 \times \ldots \times v_M}$.

Element wise we get:

$$(U \otimes U)_{u_1 \ldots u_N v_1 \ldots v_M} = U_{u_1 \ldots u_N} V_{v_1 \ldots v_M}.$$

For example, if we let

$$U = \left[\begin{array}{cc|cc} 1 & 2 & 5 & 6 \\ 3 & 4 & 7 & 8 \end{array}\right] \text{ and } V = \left[\begin{array}{cc|cc} 9 & 10 & 13 & 14 \\ 11 & 12 & 15 & 16 \end{array}\right],$$

which are two $2 \times 2 \times 2$ 3rd order tensors, then their outer product is going to be a $2 \times 2 \times 2 \times 2 \times 2 \times 2$ 6th order tensor. This I will display as a set of 3 dimensional matrices with their corresponding 4th, 5th and 6th dimensions:

$$C = \left[\begin{array}{cc|cc} 9 & 18 & 45 & 54 \\ 27 & 36 & 63 & 72 \end{array}\right]_{(1,1,1)} \left[\begin{array}{cc|cc} 11 & 22 & 55 & 66 \\ 33 & 44 & 77 & 88 \end{array}\right]_{(2,1,1)}$$

$$\left[\begin{array}{cc|cc} 10 & 20 & 50 & 60 \\ 30 & 40 & 70 & 80 \end{array}\right]_{(1,2,1)} \left[\begin{array}{cc|cc} 12 & 24 & 60 & 72 \\ 36 & 48 & 84 & 96 \end{array}\right]_{(2,2,1)} \left[\begin{array}{cc|cc} 13 & 26 & 65 & 78 \\ 39 & 52 & 91 & 104 \end{array}\right]_{(1,1,2)}$$

$$\left[\begin{array}{cc|cc} 15 & 30 & 75 & 90 \\ 45 & 60 & 105 & 120 \end{array}\right]_{(2,1,2)} \left[\begin{array}{cc|cc} 14 & 28 & 70 & 84 \\ 42 & 56 & 98 & 112 \end{array}\right]_{(1,2,2)} \left[\begin{array}{cc|cc} 16 & 32 & 80 & 96 \\ 48 & 64 & 112 & 128 \end{array}\right]_{(2,2,2)}.$$

Tensor $\times$ Tensor Inner Product The inner product is effectively the opposite of the outer product, although requires the tensors to have equal dimensions [3]. For two tensors $\in \Re^{x_1 \times x_2 \times \ldots \times x_N}$, the inner product is defined by

$$\langle A, B \rangle = A_{a_1 a_2 \ldots a_N} B_{b_1 b_2 \ldots b_N}.$$

For vectors, the inner product is defined by

$$\langle \mathbf{b}, \mathbf{c} \rangle = \mathbf{b}^T \mathbf{c}.$$

Element wise we obtain

$$\langle \mathbf{b}, \mathbf{c} \rangle = \mathbf{b}_j \mathbf{c}_i.$$

For example, if we let

$$\mathbf{b} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \text{ and } \mathbf{c} = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}$$

then their inner product is

$$\langle \mathbf{b}, \mathbf{c} \rangle = \mathbf{b}^T \mathbf{c} = \begin{bmatrix} 4 & 5 & 6 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = 32.$$

For Tensors, the inner product is the sum of the products of corresponding elements. Hence if

$$A = \left[\begin{array}{cc|cc} 1 & 2 & 5 & 6 \\ 3 & 4 & 7 & 8 \end{array}\right], B = \left[\begin{array}{cc|cc} 9 & 10 & 13 & 14 \\ 11 & 12 & 15 & 16 \end{array}\right]$$

then their inner product is

$$\langle A, B \rangle = (1 \times 9) + (2 \times 10) + \ldots + (7 \times 15) + (8 \times 16) = 492.$$

Tensor × Tensor Contracted Product The contracted product of two tensors is a generalization of the tensor × vector and tensor × matrix multiplications. The key distinction is that the modes to be multiplied and the ordering of the resulting modes is handled specially in the matrix and vector cases. In this general case, let $A \in \Re^{x_1 \times ... \times x_N \times j_1 \times ... \times j_P}$ and $B \in \Re^{x_1 \times ... \times x_N \times k_1 \times ... \times k_M}$. We can multiply both tensors along the first M modes and the result is a tensor of size $j_1 \times ... \times j_P \times k_1 \times ... \times k_M$.

Element wise we have

$$\langle A, B \rangle_{(1,...,N;1,...,N)\ j_1,...,j_P,k_1,...,k_M} = A_{x_1,...,x_N,j_1,...,j_P} B_{x_1,...,x_N,k_1,...,k_M}.$$

With this notation, the modes to be multiplied are specified in the subscripts that follow the angle brackets. The remaining modes are ordered such that those from A come before B, which is different from the tensor-matrix product case considered previously, where the leftover matrix dimension of B replaces x_n, rather than being moved to the end [3].

If we first consider two second order tensors $A \in \Re^{2\times 4}$ and $B \in \Re^{3\times 2}$, i.e. matrices. Let

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{bmatrix}, B = \begin{bmatrix} 9 & 10 \\ 11 & 12 \\ 13 & 14 \end{bmatrix}.$$

Then their contracted product $\langle A, B \rangle_{(1;2)} \in \Re^{4\times 3}$ is given by

$$\langle A, B \rangle_{(1;2)\ ij} = \sum_{k=1}^{2} A_{ki} B_{jk}$$

which gives

$$\begin{bmatrix} 59 & 71 & 83 \\ 78 & 94 & 110 \\ 97 & 117 & 137 \\ 116 & 140 & 164 \end{bmatrix}.$$

This is clearly different from that of a normal matrix multiplication encountered before, although if this method were applied to two matrices that

satisfied the normal matrix multiplication rules, the outputs would be identical.

If we now consider two 3rd order tensors $A \in \Re^{2\times3\times2}$ and $B \in \Re^{2\times3\times3}$, where

$$A = \left[\begin{array}{ccc|ccc} 1 & 2 & 3 & 7 & 8 & 9 \\ 4 & 5 & 6 & 10 & 11 & 12 \end{array}\right]$$

and

$$B = \left[\begin{array}{ccc|ccc|ccc} 13 & 14 & 15 & 19 & 20 & 21 & 25 & 26 & 27 \\ 16 & 17 & 18 & 22 & 23 & 24 & 28 & 29 & 30 \end{array}\right]$$

then there are three possible contracted products that we can calculate: $\langle A, B\rangle_{(1;1)}$, $\langle A, B\rangle_{(2;2)}$ and $\langle A, B\rangle_{(1,2;1,2)}$; two single contractions and one double contraction. If we first consider $\langle A, B\rangle_{(1;1)}$, then we will get a 4th order tensor of size $3 \times 2 \times 3 \times 3$, whose elements are defined by

$$\langle A, B\rangle_{(1;1)\ ijlm} = \sum_{k=1}^{2} A_{kij} B_{klm}$$

giving the output, where the corresponding 4th dimension is displayed next to each matrix:

$$\left[\begin{array}{cc|cc|cc} 77 & 251 & 82 & 268 & 87 & 285 \\ 106 & 280 & 113 & 299 & 120 & 318 \\ 135 & 309 & 144 & 330 & 153 & 351 \end{array}\right]_{(1)},$$

$$\left[\begin{array}{cc|cc|cc} 107 & 353 & 112 & 370 & 117 & 387 \\ 148 & 394 & 155 & 413 & 162 & 432 \\ 189 & 435 & 198 & 456 & 207 & 477 \end{array}\right]_{(2)},$$

$$\left[\begin{array}{cc|cc|cc} 137 & 455 & 142 & 472 & 147 & 489 \\ 190 & 508 & 197 & 527 & 204 & 546 \\ 243 & 561 & 252 & 582 & 261 & 603 \end{array}\right]_{(3)}.$$

If we now consider the double contraction $\langle A, B\rangle_{(1,2;1,2)} \in \Re^{2\times3}$, we calculate the elements as follows:

$$\langle A, B\rangle_{(1,2;1,2)\ ij} = \sum_{k=1}^{2}\sum_{m=1}^{3} A_{kmi} B_{kmj}$$

The output is the following 2×3 matrix:

$$\begin{bmatrix} 343 & 469 & 595 \\ 901 & 1243 & 1585 \end{bmatrix}$$

It should be noted that the inner product of two tensors is the same as the contracted product, where the answer is a scalar.

2.4 Multilinear Algebra

Multilinear algebra is a generalization of linear algebra, since a linear function is also multilinear in one variable. Multilinear algebra implies that we are dealing with functions of several variables that are linear in each variable separately. I will therefore firstly consider the algebra of matrices and extend the algebra, where possible, to tensors for multilinear algebra.

2.4.1 The Algebra of Matrices

In general I will be considering an $m \times n$ matrix:

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & ... & x_{1n} \\ x_{21} & x_{22} & x_{23} & ... & x_{2n} \\ x_{31} & x_{32} & x_{33} & ... & x_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & x_{m3} & ... & x_{mn} \end{bmatrix}$$

Addition of Matrices

Definition 2.10. *Given two $m \times n$ matrices A and B, we define the sum $C = A + B$ to be the $m \times n$ matrix whose elements are $C_{ij} = A_{ij} + B_{ij}$ [4].*

It is important to note that the sum of two matrices is defined only when they are of equal size. The sum is simply obtained by adding corresponding elements, therefore resulting in an equal sized output matrix.

Theorem 2.1. (Addition of matrices is commutative) *For two matrices A and B, both of equal size, we have $A + B = B + A$.*

Proof. From [4]. If A and B are each of size $m \times n$ then $A + B$ and $B + A$ are also of size $m \times n$ and by the above definition, element wise we have:

$$A + B = [a_{ij} + b_{ij}], \qquad B + A = [b_{ij} + a_{ij}].$$

Since the addition of numbers is commutative, we have $a_{ij} + b_{ij} = b_{ij} + a_{ij}$ for all i, j and so, we conclude that $A + B = B + A$. □

Theorem 2.2. (Addition of matrices is associative) *For the matrices A, B and C, all of equal size, we have $A + (B + C) = (A + B) + C$.*

Proof. From [4]. If A, B and C are each of size $m \times n$ then $A + (B + C)$ and $(A + B) + C$ are also of size $m \times n$. Element wise we have:

$$A + (B + C) = [a_{ij} + (b_{ij} + c_{ij})], \qquad (A + B) + C = [(a_{ij} + b_{ij}) + c_{ij}].$$

Since the addition of numbers is commutative, we have $a_{ij}+(b_{ij}+c_{ij}) = (a_{ij}+b_{ij})+c_{ij}$ for all i, j and so, we conclude that $A+(B+C) = (A+B)+C$. □

Zero and inverse matrices

Theorem 2.3. (Zero matrix) *There is a unique $m \times n$ matrix M such that, for every $m \times n$ matrix A, $A + M = A$.*

Proof. From [4]. Consider the matrix M whose entries are all zeros, i.e. $m_{ij} = 0 \quad \forall \quad i, j$. Hence element wise we have:

$$A + M = a_{ij} + m_{ij} = a_{ij} + 0 = a_{ij} = A$$

To establish the uniqueness of this matrix, assume we have another matrix B such that $A + B = A$ for every $m \times n$ matrix A. Then in particular we have $M + B = M$. But, taking B instead of A in the property for M we have $B + M = B$. It now follows by Theorem 2.1 that $B = M$. □

Definition 2.11. *The unique matrix arising in Theorem 2.3 is called the $m \times n$* ***zero matrix*** *and will be denoted simply as 0, or if confusion is caused 0_{ij}.*

Theorem 2.4. (Additive inverse) *For every $m \times n$ matrix A there is a unique matrix B such that $A + B = 0$.*

Proof. From [4]. Given $A = a_{ij}$, consider the matrix B whose elements are the additive inverse of A. Hence element wise we have:

$$A + B = a_{ij} + b_{ij} = a_{ij} + (-a_{ij}) = 0$$

To establish the uniqueness of such a matrix B, suppose there exists a matrix C such that $A+C = 0$. Then for all i, j we have $a_{ij}+c_{ij} = 0$ and consequently $c_{ij} = -a_{ij}$, which means that $C = B$. □

Definition 2.12. *The unique matrix B arising in Theorem 2.4 is called the* ***additive inverse*** *of A and will be denoted by $-A$. Thus $-A$ is the matrix whose elements are the additive inverse of the corresponding elements of A.*

Given two scalars a and b, their difference $a - b$ is defined to be $a + (-b)$. The same principle applies for matrices. A and B both of equal size. $A - B$ is written as $A+(-B)$, where A and B are of the same size. The '-' operation is also defined as matrix subtraction.

Matrix - scalar multiplication

Definition 2.13. *Given a matrix A and a scalar λ, we define their product to be the matrix λA. Whose elements are obtained by multiplying each element of A by λ, defined by $\lambda A = \lambda a_{ij}$.*

The principal properties of multiplying a matrix by a scalar are as follows.

Theorem 2.5. (Properties of multiplying a matrix by a scalar)

(1) $\lambda(A + B) = \lambda A + \lambda B$,

(2) $(\lambda + \mu)A = \lambda A + \mu A$,

(3) $\lambda(\mu A) = (\lambda\mu)A$,

(4) $(-1)A = -A$,

(5) $0A = 0_{ij}$.

Matrix - matrix multiplication Matrix multiplication has the following key properties.

Theorem 2.6. (Matrix multiplication is associative) *Given three matrices A, B and C, who have compatible sizes for multiplication, then $A(BC) = (AB)C$.*

Proof. From [4]. For $A(BC)$ to be defined we require their respective sizes to be $m \times n$, $n \times p$, $p \times q$, in which case $(AB)C$ is also defined. Computing the elements of these products we obtain:

$$[A(BC)]_{ij} = a_{ik}[BC]_{kj} = a_{ik}(b_{kt}c_{tj}) = a_{ik}b_{kt}c_{tj}$$

$$[(AB)C]_{ij} = [AB]_{it}c_{tj} = (a_{ik}b_{kt})c_{tj} = a_{ik}b_{kt}c_{tj}$$

Clearly, we can see that $A(BC) = (AB)C$. □

Theorem 2.7. (Matrix multiplication distributive law) *Given three matrices A, B and C, who have compatible sizes for multiplication, then:*

$$A(B + C) = AB + AC, \qquad (B + C)A = BA + CA$$

Proof. From [4]. For the first equality we require A to be of size $m \times n$ and B, C to be of size $n \times p$, in which case:

$$[A(B+C)]_{ij} = a_{ik}(b_{kj}+c_{kj}) = a_{ik}b_{kj}+a_{ik}c_{kj} = [AB]_{ij}+[AC]_{ij} = [AB+AC]_{ij}$$

it follows that $A(B + C) = AB + AC$. □

Theorem 2.8. (Matrix multiplication with scalars) *If AB is defined then for all scalars λ we have:*

$$\lambda(AB) = (\lambda A)B = A(\lambda B)$$

Definition 2.14. *A matrix is said to be square if it is of size $n \times n$.*

Theorem 2.9. (Identity matrix) *There is a unique $n \times n$ matrix M with the property that for every $n \times n$ matrix A, $AM = A = MA$.*

M is defined as the matrix whose elements are all zero for $i \neq j$ and 1 for $i = j$. i.e:

$$M = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}$$

This is also defined as the Kronecker delta symbol δ_{ij} which we have already encountered.

2.4.2 Fundamental Operations with Tensors

1. **Addition** The sum of two or more tensors of the same rank and type yields a tensor of the same rank and type. i.e. $A_q^{mp} + B_q^{mp} = C_q^{mp}$. Addition is commutative and associative.

2. **Subtraction** The difference of two vectors of the same rank and type is also a tensor of the same rank and type. i.e. $A_q^{mp} - B_q^{mp} = D_q^{mp}$.

3. **Contraction** If one contravariant and one covariant index of a tensor are set equal, the result indicates that a summation over the equal indices is to be taken according to the summation convention. This resulting sum is a tensor of rank two less than the original tensor. This process is called contraction. i.e. The tensor of rank 5, E_{qr}^{mps}, set $r = s$ to obtain $E_{qs}^{mps} = B_q^{mp}$, a tensor of rank 3.

4. **Outer Product** The product of two tensors is a tensor whose rank is the sum of the ranks of the given tensors. This product which involves ordinary multiplication of the components of the tensor is called the outer product. $A_q^{mp} B_r^s = E_{qr}^{mps}$.

5. **Contracted Product** By the process of outer multiplication, followed by a contraction, we obtain a new tensor called a contracted product of the given tensors. i.e. Given A_q^{mp} and B_{st}^r, the outer product is $A_q^{mp} B_{st}^r$. Letting $q = r$, we obtain the contracted product $A_r^{mp} B_{st}^r$.

6. **Inner Product** The inner product of two tensors is obtained when the contracted product of two equally sized tensors is calculated, and the product is contracted to a scalar. i.e. Given A_q^{mp} and B_{st}^r, hence letting $s = m, t = p$ and $q = r$ we get a scalar. This process is called inner multiplication.

7. **Quotient Law** Suppose it is not known if a quantity X is a tensor or not. If an inner product of X with an arbitrary tensor is itself a tensor, then X is also a tensor. This is known as the quotient law.

Addition of Tensors

Definition 2.15. *Given two $x_1 \times x_2 \times ... \times x_N$ tensors A and B, we define the sum $C = A + B$ to be the $x_1 \times x_2 \times ... \times x_N$ tensor whose elements are $C_{x_1 x_2 ... x_N} = A_{x_1 x_2 ... x_N} + B_{x_1 x_2 ... x_N}$ [4].*

Similarly to matrices, it is important to note that the sum of two tensors is defined only when they are of equal size. The sum is simply obtained by adding corresponding elements, therefore resulting in an equal sized output.

Theorem 2.10. (Addition of tensors is commutative) *For two tensors A and B, both of equal size, we have $A + B = B + A$.*

Proof. If A and B are each of size $x_1 \times x_2 \times ... \times x_N$ then $A + B$ and $B + A$ are also of size $x_1 \times x_2 \times ... \times x_N$ and by the above definition, element wise we have:

$$[A + B]_{x_1 x_2 ... x_N} = [a_{x_1 x_2 ... x_N} + b_{x_1 x_2 ... x_N}],$$

$$[B + A]_{x_1 x_2 ... x_N} = [b_{x_1 x_2 ... x_N} + a_{x_1 x_2 ... x_N}]$$

Since the addition of numbers is commutative, we have $a_{x_1 x_2 ... x_N} + b_{x_1 x_2 ... x_N} = b_{x_1 x_2 ... x_N} + a_{x_1 x_2 ... x_N}$ for all $x_1, x_2, ..., x_N$ and so we conclude that $A + B = B + A$. □

Theorem 2.11. (Addition of tensors is associative) *For the tensors A, B and C, all of equal size, we have $A + (B + C) = (A + B) + C$.*

Proof. If A, B and C are each of size $x_1 \times x_2 \times ... \times x_N$ then $A + (B + C)$ and $(A + B) + C$ are also of size $x_1 \times x_2 \times ... \times x_N$. Element wise we have:

$$A + (B + C) = [a_{x_1 x_2 ... x_N} + (b_{x_1 x_2 ... x_N} + c_{x_1 x_2 ... x_N})],$$

$$(A + B) + C = [(a_{x_1 x_2 ... x_N} + b_{x_1 x_2 ... x_N}) + c_{x_1 x_2 ... x_N}].$$

Since the addition of numbers is commutative, we have $a_{x_1 x_2 ... x_N} + (b_{x_1 x_2 ... x_N} + c_{x_1 x_2 ... x_N}) = (a_{x_1 x_2 ... x_N} + b_{x_1 x_2 ... x_N}) + c_{x_1 x_2 ... x_N}$ for all $x_1, x_2, ..., x_N$ and so we conclude that $A + (B + C) = (A + B) + C$. □

Zero and inverse tensors

Theorem 2.12. (Zero tensor) *There is a unique $x_1 \times x_2 \times ... \times x_N$ tensor T such that, for every $x_1 \times x_2 \times ... \times x_N$ tensor A, $A + T = A$.*

Proof. Consider the tensor T whose entries are all zeros, i.e. $t_{x_1 x_2 ... x_N} = 0 \quad \forall \quad x_1, x_2, ..., x_N$, hence element wise we have

$$A + T = a_{x_1 x_2 ... x_N} + t_{x_1 x_2 ... x_N} = a_{x_1 x_2 ... x_N} + 0 = a_{x_1 x_2 ... x_N} = A$$

To establish the uniqueness of this tensor, assume we have another tensor S such that $A + S = A$ for every $x_1 \times x_2 \times ... \times x_N$ tensor A. Then in particular we have $T + S = T$. But, taking S instead of A in the property for T we have $S + T = S$. It now follows by Theorem 2.10 that $S = T$. Hence unique. □

Definition 2.16. *The unique tensor arising in Theorem 2.12 is called the $x_1 \times x_2 \times ... \times x_N$* ***zero tensor*** *and will be denoted simply as 0, or if confusion is caused $0_{x_1 x_2 ... x_N}$.*

Theorem 2.13. (Additive inverse of a tensor) *For every $x_1 \times x_2 \times ... \times x_N$ tensor A there is a unique tensor B such that $A + B = 0$.*

Proof. Given $A = a_{x_1 x_2 ... x_N}$, consider the tensor B whose elements are the additive inverse of A. Hence element wise we have:

$$A + B = a_{x_1 x_2 ... x_N} + b_{x_1 x_2 ... x_N} = a_{x_1 x_2 ... x_N} + (-a_{x_1 x_2 ... x_N}) = 0$$

To establish the uniqueness of such a tensor B, suppose there exists a tensor C such that $A + C = 0$. Then for all $x_1, x_2, ..., x_N$ we have $a_{x_1 x_2 ... x_N} + c_{x_1 x_2 ... x_N} = 0$ and consequently $c_{x_1 x_2 ... x_N} = -a_{x_1 x_2 ... x_N}$, which means that $C = B$. Hence unique. □

Definition 2.17. *The unique tensor B arising in Theorem 2.13 is called the* ***additive inverse*** *of A and will be denoted by $-A$. Thus $-A$ is the tensor whose elements are the additive inverse of the corresponding elements of A.*

As with matrices, we defined their difference as $A + (-B)$. This is the same for tensors.

Tensor - scalar multiplication

Definition 2.18. *Given a tensor A and a scalar λ, we define their product to be the tensor λA. Whose elements are obtained by multiplying each element of A by λ, defined by $\lambda A = \lambda a_{x_1 x_2 \ldots x_N}$.*

The principal properties of multiplying a tensor by a scalar are the same as that of multiplying a matrix by a scalar, as seen in Theorem 2.5.

Tensor multiplication

Theorem 2.14. (Tensor inner product commutative) *Given two equally sized tensors A and B then there inner product is defined by $\langle A, B\rangle = \langle B, A\rangle$.*

Proof. Given $A = a_{x_1 x_2 \ldots x_N}$ and $b_{x_1 x_2 \ldots x_N}$, and using the fact that multiplication of scalars is commutative:

$$\langle A, B\rangle = a_{x_1 x_2 \ldots x_N} b_{x_1 x_2 \ldots x_N} = b_{x_1 x_2 \ldots x_N} a_{x_1 x_2 \ldots x_N} = \langle B, A\rangle$$

□

The outer product of tensors isn't commutative but is associative.

Theorem 2.15. (Tensor Outer product associative) *Given three tensors A, B and C of sizes $x_1 \times \ldots \times x_N$, $y_1 \times \ldots \times y_M$ and $v_1 \times \ldots \times v_K$ respectively then:*

$$(A \otimes B) \otimes C = A \otimes (B \otimes C)$$

Proof.

$$[(A \otimes B) \otimes C]_{x_1,\ldots,x_N,y_1,\ldots,y_M,v_1,\ldots,v_K} = [A \otimes B]_{x_1,\ldots,x_N,y_1,\ldots,y_M} c_{v_1,\ldots,v_K}$$

$$= a_{x_1,\ldots,x_N} b_{y_1,\ldots,y_M} c_{v_1,\ldots,v_K}$$

$$[A \otimes (B \otimes C)]_{x_1,\ldots,x_N,y_1,\ldots,y_M,v_1,\ldots,v_K} = a_{x_1,\ldots,x_N} [B \otimes C]_{y_1,\ldots,y_M,v_1,\ldots,v_K}$$

$$= a_{x_1,\ldots,x_N} b_{y_1,\ldots,y_M} c_{v_1,\ldots,v_K}$$

□

The contracted product, which is effectively a generalised version of matrix multiplication, also holds for associativity.

Theorem 2.16. (Tensor contracted product associative) *Given three tensors A, B and C with sizes $x_1 \times x_2 \times ... \times x_i \times ...x_N$, $x_1 \times x_2 \times ... \times x_i \times ... \times x_j \times ...x_N$ and $x_1 \times x_2 \times ... \times x_j \times ...x_N$ respectively then:*

$$\left\langle \langle A, B\rangle_{(x_i;x_i)}, C\right\rangle_{(x_j;x_j)} = \left\langle A, \langle B, C\rangle_{(x_j;x_j)}\right\rangle_{(x_i;x_i)}$$

Proof.

$$\begin{aligned}\left\langle \langle A, B\rangle_{(x_i;x_i)}, C\right\rangle_{(x_j;x_j)} &= \langle A_{x_1x_2...k...x_N} B_{x_1x_2...k...x_j...x_N}, C\rangle_{(x_j;x_j)} \\ &= A_{x_1x_2...k...x_N} B_{x_1x_2...k...p...x_N} C_{x_1x_2...p...x_N}\end{aligned}$$

$$\begin{aligned}\left\langle A, \langle B, C\rangle_{(x_j;x_j)}\right\rangle_{(x_i;x_i)} &= \langle A, B_{x_1x_2...x_i...p...x_N} C_{x_1x_2...p...x_N}\rangle_{(x_i;x_i)} \\ &= A_{x_1x_2...k...x_N} B_{x_1x_2...k...p...x_N} C_{x_1x_2...p...x_N}\end{aligned}$$

□

2.5 Tensor Rank and Decomposition

Definition 2.19. ***Rank-one tensors.*** *An N-way tensor $X \in \Re^{x_1 \times x_2 \times ... \times x_N}$ is a rank one tensor if it can be written as the outer product of N vectors, i.e:*

$$X = a^{(1)} \otimes a^{(2)} \otimes ... \otimes a^{(N)}$$

This means that each element of the tensor is the product of the corresponding vector elements [9].

For example:

$$\left[\begin{array}{ccc|ccc|ccc} 28 & 35 & 42 & 32 & 40 & 48 & 36 & 45 & 54 \\ 56 & 70 & 84 & 64 & 80 & 96 & 72 & 90 & 108 \\ 84 & 105 & 126 & 96 & 120 & 144 & 108 & 135 & 162 \end{array}\right] = \left[\begin{array}{c} 1 \\ 2 \\ 3 \end{array}\right] \otimes \left[\begin{array}{ccc} 4 & 5 & 6 \end{array}\right] \otimes \left[\begin{array}{c|c|c} 7 & 8 & 9 \end{array}\right]$$

Definition 2.20. ***Tensor Rank*** *The rank of a tensor X, denoted rank(X), is defined as the smallest number of rank-one tensors that generate X as their sum [9]. Hence the rank of X is R for:*

$$X = \sum_{i=1}^{R} a_i^{(1)} \otimes a_i^{(2)} \otimes ... \otimes a_i^{(N)}$$

If you consider the above example of a rank one tensor, but alter the (1,1,1) element from 28 to 30, then the tensor is no longer of rank 1. It is now of rank 2:

$$\left[\begin{array}{ccc|ccc|ccc} 30 & 35 & 42 & 32 & 40 & 48 & 36 & 45 & 54 \\ 56 & 70 & 84 & 64 & 80 & 96 & 72 & 90 & 108 \\ 84 & 105 & 126 & 96 & 120 & 144 & 108 & 135 & 162 \end{array}\right]$$

$$= \left[\begin{array}{ccc|ccc|ccc} 28 & 35 & 42 & 32 & 40 & 48 & 36 & 45 & 54 \\ 56 & 70 & 84 & 64 & 80 & 96 & 72 & 90 & 108 \\ 84 & 105 & 126 & 96 & 120 & 144 & 108 & 135 & 162 \end{array}\right]$$

$$+ \left[\begin{array}{ccc|ccc|ccc} 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

$$= \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \otimes \left[\begin{array}{ccc} 4 & 5 & 6 \end{array}\right] \otimes \left[\begin{array}{c|c|c} 7 & 8 & 9 \end{array}\right]$$

$$+ \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix} \otimes \left[\begin{array}{ccc} 1 & 0 & 0 \end{array}\right] \otimes \left[\begin{array}{c|c|c} 1 & 0 & 0 \end{array}\right]$$

Unfortunately there is no straight forward algorithm to determine the rank of a specific given tensor. In practice, the rank of a tensor is determined numerically by fitting various rank-R CANDECOMP/PARAFAC models [12].

Kruskal [10] discusses the case of $2 \times 2 \times 2$ tensors which have typical ranks of two and three over $\Re$. In fact, Monte Carlo experiments (which randomly draw each entry of the tensor from a normal distribution with mean zero and standard deviation one) reveal that the set of $2 \times 2 \times 2$ tensors of rank two fills about 79% of the space while those of rank three fill 21%. Rank-one tensors are possible but occur with almost zero probability [9].

2.5.1 The CP (CANDECOMP/PARAFAC) decomposition

As mentioned previously, there is no finite algorithm to determine the rank of a tensor, the most popular model used in determining a tensors decomposition is the CP decomposition, from [9] defined as:

$$A_{x_1 x_2 \ldots x_d} = \sum_{r=1}^{R} a_{1(x_1 r)} a_{2(x_2 r)} \ldots a_{d(x_d r)}$$

The first issue that arises in computing the CP decomposition is how to choose the number R, of rank one components. One technique is to fit multiple CP decompositions with differing numbers of rank one components until one is "good". We could also calculate the CP decomposition for 1,2,3,... rank one components until we get the correct solution, however, there are many problems with this procedure. As "real life" data is often very large, chaotic and noisy. We may have approximations of a lower rank that are arbitrarily close in terms of fit, but cause problems in practice compared to the approximations at a higher rank.

2.5.2 The TT-decomposition

Recently a new decomposition, named the tensor train, or TT-decomposition, was proposed for compact representation and approximation of high-dimensional tensors. It can be computed via standard decompositions (such as the SVD and QR) but does not suffer from the curse of dimensionality. By the curse of dimensionality we mean that the memory required to store an array with d indices and the amount of operations required to perform basic operations with such an array grows exponentially in the dimensionality d. Therefore, direct manipulation of general d-dimensional arrays seems to be impossible. The TT-decomposition is written as a tensor train of the form:

$$A(i_1, i_2, ..., i_d) \approx \sum_{\alpha_1, \ldots, \alpha_d} G_1(i_1, \alpha_1) G_2(\alpha_1, i_2, \alpha_2) \ldots G_{d-1}(\alpha_{d-2}, i_{d-1}, \alpha_{d-1}) G_d(\alpha_{d-1}, i_d)$$

With tensor carriages $G_1, ..., G_d$, where any two neighbours have a common summation index. The summation indices α_k run from 1 to r_k (r_k is the compression rank) and are referred to as auxiliary indices, in contrast to the initial indices i_k that are called spacial indices. The tensor carriages G_k have sizes $r_k - 1 \times n_k \times r_k + 1$ except for $k = 1$ and $k = d$ where they have sizes

$n_1 \times r_1$ and $r_d - 1 \times n_d$, respectively. It is sometimes convenient to assume that G_1 and G_d are not two dimensional but in fact three-dimensional with sizes $1 \times n_1 \times r_1$ and $r_d - 1 \times n_d \times 1$ [12].

Theorem 2.17. (TT-decomposition [12])
For any tensor $A = [A(i_1, ..., i_d)]$ there exists a TT approximation $T = [T(i_1, ..., i_d)]$ with compression ranks r_k such that:

$$\| A - T \|_F \leq \sqrt{\sum_{k=1}^{d-1} \epsilon_k^2}$$

where ϵ_k is the distance (in the Frobenious norm) from A_k to its best rank-r_k approximation:

$$\epsilon_k = \min_{rank\, B \leq r_k} \| A_k - B \|_F$$

The proof of which can be found in [12], which gives us the following algorithm.

TT-decomposition algorithm. Given a tensor $A \in \Re^{n_1 \times n_2 \times ... \times n_d}$, and an accuracy bound ϵ.

1. Compute the Frobenious norm $nrm = A$,
2. Compute the size of the first unfolded matrix A_1:

$$N_l = n_1, N_r = \Pi_{k=2}^{d} n_k$$

3. Create a temporary tensor $M = A$,
4. Unfold the tensor into the calculated dimensions $M = M \in \Re^{N_l \times N_r}$,
5. Compute the truncated SVD of $M \approx USV$ so that the rank r satisfies:

$$\sqrt{\sum_{k=r+1}^{min(N_l, N_r)} \sigma_k^2} \leq \frac{\epsilon \cdot nrm}{\sqrt{d-1}}$$

6. Set the first carriage $G_1 = U$, recalculate $M = SV^T = (VS)^T$ and let $r_1 = r$,

7. Process the remaining modes from $k = 2$ to $d - 1$,

8. Calculate the dimensions of the next carriage:

$$N_l = n_k, N_r = \frac{N_r}{n_k}$$

9. Unfold to the correct size $M = M \in \Re^{rN_l \times N_r}$,

10. Compute the truncated SVD of $M \approx USV$ so that the rank r satisfies:

$$\sqrt{\sum_{k=r+1}^{min(N_l,N_r)} \sigma_k^2} \leq \frac{\epsilon \cdot nrm}{\sqrt{d-1}}$$

11. Reshape U into the k^{th} carriage:

$$G_k = U \in \Re r_{k-1} \times n_k \times r_k$$

12. Recalculate $M = SV$,

13. Once the first $d - 1$ carriage's have been calculated, $G_d = M^T$.

2.5.3 Computing the TT-decomposition

To demonstrate the TT-decomposition algorithm consider the tensor $A \in \Re^{2\times2\times2}$ and error 0.1:

$$\left[\begin{array}{cc|cc} 1 & 2 & 5 & 6 \\ 3 & 4 & 7 & 8 \end{array}\right]$$

The Frobenious norm of A, $\| A \|_F = 14.2829$,

The error bound we acquire with $d = 3$ is 1.01,

The first unfolding M is of size $2 \times (2 \times 2) = 2 \times 4$:

$$M = \left[\begin{array}{cccc} 1 & 2 & 5 & 6 \\ 3 & 4 & 7 & 8 \end{array}\right]$$

Computing the SVD of M gives

$$U = \begin{bmatrix} -0.5667 & -0.8239 \\ -0.8239 & 0.5667 \end{bmatrix}$$

$$S = \begin{bmatrix} 14.2358 & 0 & 0 & 0 \\ 0 & 1.1585 & 0 & 0 \end{bmatrix}$$

$$V = \begin{bmatrix} -0.2134 & 0.7564 & -0.4314 & -0.4430 \\ -0.3111 & 0.5344 & 0.3905 & 0.6820 \\ -0.6042 & -0.1316 & 0.5952 & -0.5133 \\ -0.7019 & -0.3536 & -0.5542 & 0.2742 \end{bmatrix}$$

Hence, from S we can see that the rank of M, $r_1 = 2$.

Due to $r_1 = 2$ we must adjust the U, S, V matrices to contain orthonormal columns of U, 2 orthonormal rows of V^T and 2 orthonormal columns and rows of S:

$$U = \begin{bmatrix} -0.5667 & -0.8239 \\ -0.8239 & 0.5667 \end{bmatrix}$$

$$S = \begin{bmatrix} 14.2358 & 0 \\ 0 & 1.1585 \end{bmatrix}$$

$$V = \begin{bmatrix} -0.2134 & 0.7564 \\ -0.3111 & 0.5344 \\ -0.6042 & -0.1316 \\ -0.7019 & -0.3536 \end{bmatrix}$$

We now set $G_1 = U$ and recalculate $M = (VS)^T$:

$$G_1 = \begin{bmatrix} -0.5667 & -0.8239 \\ -0.8239 & 0.5667 \end{bmatrix}$$

$$M = \begin{bmatrix} -3.0384 & -4.4291 & -8.6010 & -9.9916 \\ 0.8763 & 0.6191 & -0.1525 & -0.4097 \end{bmatrix}$$

Reshape to size $M \in \Re^{rN_l \times N_r} \equiv (2 \times 2) \times 2 = 4 \times 2$:

$$M = \begin{bmatrix} -3.0384 & -8.6010 \\ 0.8763 & -0.1525 \\ -4.4291 & -9.9916 \\ 0.6191 & -0.4097 \end{bmatrix}$$

We now compute the SVD of M to get

$$U = \begin{bmatrix} -0.6405 & -0.3341 & -0.6835 & -0.1046 \\ 0.0132 & -0.6914 & 0.4160 & -0.5906 \\ -0.7678 & 0.2746 & 0.5750 & 0.0664 \\ -0.0103 & -0.5788 & 0.1705 & 0.7974 \end{bmatrix}$$

$$S = \begin{bmatrix} 14.2274 & 0 \\ 0 & 1.2573 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$V = \begin{bmatrix} 0.3762 & -0.9266 \\ 0.9266 & 0.3762 \end{bmatrix}$$

Here M is of rank 2, and we again adjust the U, S, V matrices as such:

$$U = \begin{bmatrix} -0.6405 & -0.3341 \\ 0.0132 & -0.6914 \\ -0.7678 & 0.2746 \\ -0.0103 & -0.5788 \end{bmatrix}$$

$$S = \begin{bmatrix} 14.2274 & 0 \\ 0 & 1.2573 \end{bmatrix}$$

$$V = \begin{bmatrix} 0.3762 & -0.9266 \\ 0.9266 & 0.3762 \end{bmatrix}$$

We now set $G_2 = U \in \Re^{r_1 \times n_2 \times r_2} \equiv 2 \times 2 \times 2$ and recalculate $M = SV$:

$$G_2 = \left[\begin{array}{cc|cc} -0.6405 & -0.7678 & -0.3341 & 0.2746 \\ 0.0132 & -0.0103 & -0.6914 & -0.5788 \end{array}\right]$$

$$M = \begin{bmatrix} 5.3519 & 13.1824 \\ -1.1650 & 0.4730 \end{bmatrix}$$

Now we must finally calculate $G_3 = M^T$:

$$G_3 = \begin{bmatrix} 5.3519 & -1.1650 \\ 13.1824 & 0.4730 \end{bmatrix}$$

Hence

$$A \approx G_1 G_2 G_3$$

$$\left[\begin{array}{cc|cc} 1 & 2 & 5 & 6 \\ 3 & 4 & 7 & 8 \end{array}\right]$$

$$\approx \begin{bmatrix} -0.5667 & -0.8239 \\ -0.8239 & 0.5667 \end{bmatrix} \left[\begin{array}{cc|cc} -0.6405 & -0.7678 & -0.3341 & 0.2746 \\ 0.0132 & -0.0103 & -0.6914 & -0.5788 \end{array}\right] \begin{bmatrix} 5.3519 & -1.1650 \\ 13.1824 & 0.4730 \end{bmatrix}$$

$$= \left[\begin{array}{cc|cc} 1.0002 & 1.9998 & 5.0005 & 5.9996 \\ 3.0001 & 4.0000 & 6.9999 & 8.0000 \end{array}\right]$$

Definition 2.21. ***A Hilbert tensor*** *is a d dimensional tensor of size $i_1 \times i_2 \times ... \times d$ and contains the elements*

$$A_{i_1 i_2 ... i_d} = \frac{1}{i_1 + i_2 + ... + i_d}$$

Hence the 2-dimensional Hilbert tensor of size 2×2 is

$$\begin{bmatrix} 1/2 & 1/3 \\ 1/3 & 1/4 \end{bmatrix}$$

The 3-dimensional Hilbert tensor of size $3 \times 3 \times 3$ is

$$\left[\begin{array}{ccc|ccc|ccc} 1/3 & 1/4 & 1/5 & 1/4 & 1/5 & 1/6 & 1/5 & 1/6 & 1/7 \\ 1/4 & 1/5 & 1/6 & 1/5 & 1/6 & 1/7 & 1/6 & 1/7 & 1/8 \\ 1/5 & 1/6 & 1/7 & 1/6 & 1/7 & 1/8 & 1/7 & 1/8 & 1/9 \end{array}\right]$$

The 3-dimensional Hilbert tensor of size $5 \times 2 \times 2$ is

$$\left[\begin{array}{cc|cc} 1/3 & 1/4 & 1/4 & 1/5 \\ 1/4 & 1/5 & 1/5 & 1/6 \\ 1/5 & 1/6 & 1/6 & 1/7 \\ 1/6 & 1/7 & 1/7 & 1/8 \\ 1/7 & 1/8 & 1/7 & 1/9 \end{array}\right]$$

etc.

2.5.4 Numerical Experiments

I will now run the TT-decomposition algorithm on random dense tensors, random sparse tensors and Hilbert tensors of increasing dimensions to compare accuracy and time taken (in seconds) to complete. In all of these tensors each dimension will be of size 3, and so a d-dimensional tensor will have 3^d elements.

As the algorithm requires the use of the singular value decomposition, I will only be able to compute up to, and including 9 dimensions. This is due to an unfolded 10 dimensional tensor requires more memory to compute than MATLAB has.

To calculate the absolute error, we recompute the tensor from the tensor train to get the approximate solution and then sum the absolute differences of all the corresponding elements.

Dimensions	Time(s)	Absolute Error
3	0.000599	5.481726184086710e-015
4	0.000950	3.160249839595508e-013
5	0.001400	3.229430611817463e-013
6	0.003586	1.219748260727904e-012
7	0.012798	6.101206345698884e-012
8	0.190649	3.496403297095774e-011
9	1.868366	4.497736081146564e-011

Table 1: Dense Tensor with accuracy 0.00001.

Dimensions	Time(s)	Absolute Error
3	0.000595	5.481726184086710e-015
4	0.000949	3.160249839595508e-013
5	0.001390	3.229430611817463e-013
6	0.003524	1.219748260727904e-012
7	0.012970	6.101206345698884e-012
8	0.191436	3.496403297095774e-011
9	1.868051	4.497736081146564e-011

Table 2: Dense Tensor with accuracy 0.001.

Tables 1,2,3 show the time taken and accuracy of the TT-decomposition algorithm upon randomly generated tensors. Although the different levels of accuracy were applied to the same tensor, as to help comparing results.

You can see that with errors 0.00001 and 0.001 in tables 1 and 2 identical results are obtained. As expected, the times taken grow exponentially as the dimensions increase, due to a d dimensional tensor having 3^d elements. Although it only requires 1.87 seconds to compute a 9-dimensional tensor which contains 19683 elements, and the absolute error is 4.4977e-011.

Dimensions	Time(s)	Absolute Error
3	0.000462	0.548172618408671
4	0.000922	1.222504139360485
5	0.001388	3.229430611817463
6	0.003230	11.97544192366226
7	0.010927	36.17063990114404
8	0.189168	103.0903731752137
9	1.783017	409.5933577147941

Table 3: Dense Tensor with accuracy 0.1.

When the accuracy was lowered to 0.1, table 3, the times taken to complete the algorithm improve, although not greatly. The cost of this improved time is a vastly increased absolute error, where the 9-dimensional tensor now has an error of 409.59, which is only an average element error of 0.0208, but is far worse than with accuracy levels 0.001 and 0.00001.

Dimensions	Time(s)	Absolute Error
3	0.000608	2.664535259100376e-015
4	0.000856	8.658229781327576e-007
5	0.001238	3.633284856321972e-006
6	0.002217	9.628497013636217e-006
7	0.009952	2.162607546331297e-005
8	0.169623	1.059060077813001e-003
9	1.765740	3.356218074425516e-003

Table 4: Hilbert Tensor with accuracy 0.00001.

Tables 4,5 show the time taken and accuracy of the TT-decomposition algorithm upon Hilbert tensors.

If we compare the results of the Hilbert tensors, which have an ordered structure, too that of the random tensors, the random tensors gave much more accurate results. In fact, to achieve a similar error with the 9-dimensional Hilbert tensor to that of the random tensor we must increase the accuracy level to 0.0000000000001, which takes the same amount of time to compute. This is most likely down to the random tensors having a much higher rank

Dimensions	Time(s)	Absolute Error
3	0.000586	0.003892207201136
4	0.000798	0.005054637681228
5	0.001130	0.012985484130547
6	0.002040	0.023844744642908
7	0.009184	0.046796732618198
8	0.160419	0.094324369187404
9	1.648268	0.201164134157645

Table 5: Hilbert Tensor with accuracy 0.001.

to that of the Hilbert tensors.

Table 5 has the most efficient times to compute, which of course leads to an increased error. Although a much smaller one than observed with the random tensors and a accuracy level of 0.1.

Dimensions	Time(s)	Absolute Error
3	0.000581	4.506840310874860e-015
4	0.000956	3.698752148568742e-014
5	0.001253	5.314365469582339e-013
6	0.003487	3.219748260727904e-012
7	0.013145	2.1684692243965474-012
8	0.200458	7.1625748959623314-011
9	1.752219	3.563275864215860e-011

Table 6: Sparse Tensor with accuracy 0.00001.

The decomposition of sparse tensors as seen in tables 6 and 7 , which I generated using my sparse tensor MATLAB program.

Similarly to the random tensors, errors of 0.00001 and 0.001 give almost identical results. Again the times taken grow exponentially as the dimensions increase, due to a d dimensional tensor having 3^d elements.

When the accuracy was lowered from 0.00001 to 0.001, the times taken to complete the algorithm change very little. Although the absolute error is

Dimensions	Time(s)	Absolute Error
3	0.000582	4.7582758063489e-015
4	0.000954	3.1688198340326e-013
5	0.001218	2.6846912027582e-013
6	0.003389	1.7586692758790e-012
7	0.013102	6.1684695698884e-011
8	0.189921	5.4964032427583e-011
9	1.698478	3.27586120634325-011

Table 7: Sparse Tensor with accuracy 0.001.

almost identical and so it would be of interest to use a smaller error level.

2.5.5 Results

Increasing the accuracy level decreases the absolute error, with only a small effect on the time taken to compute the decomposition.

2.5.6 Applications of the TT-decomposition

The TT-decomposition has many potential applications including high-dimensional integration and multivariate function approximation. I will be focusing on the high-dimensional integration in the next section. Multivariate function approximation has a wide scope of applications in such areas as telecommunications, as it provides the ability to compress a large amount of data to a portion of the size. This will improve the efficiency of moving the data around, and then the data can be reconstructed easily.

3 High-dimensional integration using the TT-decomposition

The TT-decomposition algorithm can be applied to high-dimensional integration [12], suppose we have a function f of d variables and are required to calculate a d-dimensional integral of the form:

$$I(f) = \int_{[0,1]^d} f(x_1, x_2, ..., x_d) dx_1 dx_2 ... dx_d$$

In order to evaluate the integral we must construct a quadratic rule, consider a one dimensional rule over n points:

$$\int_0^1 g(x)dx \approx \sum_{k=1}^{n} w_k g(x_k)$$

Where there are many different quadrature schemes to determine the weights. We can extend this to a d-dimensional quadrature rule as the tensor product of one-dimensional rules:

$$I(f) \approx Q(f) = \sum_{k_1, k_2, ..., k_d} f(x_{k_1}, x_{k_2}, ..., x_{k_d}) w_{k_1} w_{k_2} ... w_{k_d}$$

This approximate integration rule reduces to the mode contractions of tensor by vector multiplications:

$$I(f) \approx Q(f) = A \times_1 w \times_2 ..... \times_d w$$

The right hand side of which can be calculated in $\mathcal{O}(dnr^3)$ operations by the algorithm given in [11]. However, we can reduce the required number of operations to $\mathcal{O}(dnr^2)$ by substituting the tensor A for its TT-decomposition with carriages $G_1, ..., G_d$:

$$Q(f) = \sum_{i_1, ..., i_d} \sum_{\alpha_1, ..., \alpha_{d-1}} G_1(i_1, \alpha_1) ... G_d(\alpha_d, i_d) w(i_1) ... w(i_d)$$

TT Contraction algorithm. Given a tensor $A \in \Re^{n_1 \times n_2 \times ... \times n_d}$ which is in the TT format with carriages G_k; and vectors w_k of size n_k.

1. $v = G_1 \times_1 w_1$,
2. for $k = 2$ to d compute the remaining modes,
3. Summate over α_{k-1}:
4. $W = G_k \times_1 v$, where W is $n_k \times r_k$,
5. Summate over i_k:
6. $v = W^T w_k$, where v is of size r_k,
7. end for loop,
8. $I = v$ where $I = A \times_1 w_1 \times_2 ... \times_d w_d$

3.1 Computation of high-dimensional integration

3.1.1 Example 1

I will firstly consider the sine example as follows:

$$f(x_1, x_2, ..., x_d) = \sin(x_1 + x_2 + ... + x_d) \tag{15}$$

The compression ranks in this case are equal to 2 [12], of course, over a complex field the canonical rank is 2 due to the identity:

$$\sin(x) = \frac{e^{ix} - e^{-ix}}{2i}$$

and that gives an exact value of the integral:

$$I(d) = Im \int_{[0,1]^d} e^{i(x_1 + x_2 + ... + x_d)} dx_1 dx_2 ... dx_d$$

$$
\begin{aligned}
&= Im \int_{[0,1]^d} e^{ix_1} e^{ix_2} ... e^{ix_d} dx_1 dx_2 ... dx_d \\
&= Im \int_{[0,1]^{d-1}} e^{ix_1} e^{ix_2} ... e^{ix_{d-1}} \frac{e^{ix_d}}{i} dx_1 dx_2 ... dx_{d-1} \\
&= ... \\
&= Im \left(\left[\frac{e^{ix_1}}{i} \frac{e^{ix_2}}{i} ... \frac{e^{ix_d}}{i} \right]_0^1 \right) \\
&= Im \left(\left(\frac{e^i}{i} - \frac{1}{i} \right)^d \right) \\
&= Im \left(\left(\frac{e^i - 1}{i} \right)^d \right) \\
&= Im \left(\frac{(e^i - 1)^d}{i^d} \right) \\
&= Im \left(\frac{i^d (e^i - 1)^d}{(-1)^d} \right) \\
&= Im \left(\frac{i^d (\cos(1) + i \sin(1) - 1)^d}{(-1)^d} \right) \\
&= Im \left(-(i(\cos(1) - 1) - \sin(1))^d \right)
\end{aligned}
$$

Therefore for $d = 1$, $I(1) = 1 - \cos(1)$,

$d = 2$, $I(2) = -2 \sin(1)(\cos(1) - 1)$,

$d = 3$, $I(3) = (\cos(1) - 1)(\cos(1) - 1 - 3 \sin^2(1))$, etc.

I will now apply the algorithm to the example shown in equation 15, using Gaussian quadrature with varying step sizes to compare the absolute errors and the efficiency. The step size is determined by the sizes of each dimension, for example a 3-dimensional tensor of size $3 \times 3 \times 3$ will have a step size of 0.5, due to integrating from 0 to 1 and taking 3 steps in each dimension. A 3-dimensional tensor of size $5 \times 5 \times 5$ will have a step size of 0.25 due to integrating from 0 to 1 and taking 5 steps in each dimension. It is possible

to take different step sizes in different dimension, but I will be consistent and keep them equal, hence for $A \in \Re^{x_1 \times x_2 \times ... \times x_d}, x_1 = x_2 = ... = x_d$.

The Gaussian quadrature Formula for one-dimensional integration is as follows:

$$\int_a^b f(x)dx = \frac{b-a}{2} \int_{-1}^{1} f\left(\frac{b-a}{2}x + \frac{b+a}{2}\right) dx$$

$$\approx \frac{b-a}{2} \sum_{k=1}^{n} w_i f\left(\frac{b-a}{2}x_i + \frac{b+a}{2}\right)$$

$$\text{where} \quad w_i = \frac{2}{(1-x_i^2)(P'_n(x_i))^2}$$

We can extend this to a multidimensional problem:

$$\int_a^b ... \int_a^b f(x_1, ..., x_d)dx_1...dx_d$$

$$= \left(\frac{b-a}{2}\right)^d \int_{-1}^{1} ... \int_{-1}^{1} f\left(\frac{b-a}{2}x_1 + \frac{b+a}{2}, ..., \frac{b-a}{2}x_d + \frac{b+a}{2}\right) dx_1...dx_d$$

$$\approx \left(\frac{b-a}{2}\right)^d \sum_{k_1=1}^{n} ... \sum_{k_d=1}^{n} w_{k_1}...w_{k_d} f\left(\frac{b-a}{2}(x_{k_1} + ... + x_{k_d}) + d\left(\frac{b+a}{2}\right)\right)$$

$$\text{where} \quad w_{k_i} = \frac{2}{(1-x_{k_i}^2)(P'_n(x_{k_i}))^2}$$

For the integration problem stated above, the associated polynomials are Legendre polynomials, $P_n(x)$. With the nth polynomial normalized to give $P_n(1) = 1$, the ith Gauss node, x_i, is the ith root of P_n [1]. The following, table 8 gives the numerical values for the first few points and weights.

Table 8: Abscissae and weights of Gaussian quadrature

Number of points	x_i	w_i
2	-0.57735	1
	$+0.57735$	1
3	-0.77459	0.555556
	0	0.888889
	$+0.77459$	0.555556
4	-0.86114	0.347855
	-0.33998	0.652145
	$+0.33998$	0.652145
	$+0.86114$	0.347855
5	-0.9061798	0.236927
	-0.5384693	0.478629
	0	0.568889
	$+0.5384693$	0.478629
	$+0.9061798$	0.236927

Example 1 in 3-dimensions. Tables 9 and 10 display the numerical solutions to equation 15 with decreasing step size's from the TT-decomposition and tensor respectively, in 3-dimensions.

steps (step size)	Q	Absolute Error	time(s)
2 (0.5)	0.878724503604261	6.304270411398028e-004	0.000373
3 ($0.3\dot{3}$)	0.879356284977070	1.354331669478626e-006	0.000374
4 (0.25)	0.879354929107617	1.537783456129205e-009	0.000381
5 (0.2)	0.879354930646481	1.080580069867665e-012	0.000368

Table 9: Example 1, in 3 dimensions, from the TT-decomposition

The exact solution for $I(d)$ where d=3 is 0.879354930645401, and even for the smallest step size of 0.5 with three points we get a good approximation with an error of $\times 10^{-4}$. When we reach a smaller step size of 0.2, which is calculated with 6 points, we acquire a solution correct to 11 decimal places, which is extremely accurate.

Another observation that can be made is that as the step size decreases,

steps (step size)	Q	Absolute Error	time(s)
2 (0.5)	0.878724503604261	6.304270411401358e-004	0.000375
3 (0.3$\dot{3}$)	0.879356284977070	1.354331668923514e-006	0.000371
4 (0.25)	0.879354929107617	1.537783678173810e-009	0.000379
5 (0.2)	0.879354930646481	1.079913936052890e-012	0.000381

Table 10: Example 1, in 3 dimensions, from the tensor

the tensor gives a slightly better result to that of the TT-decomposition. This may be due to the error generated from decomposing the tensor into a tensor train.

Also the time taken to compute different step sizes are all very similar and there is no difference between the computation via the tensor or the TT-decomposition. The computation via the tensor is much more efficient though, as in order to compute the contraction via the TT-decomposition, you must ofcourse first compute the TT-decomposition. This also limits the number of dimensions that can be investigated, as the TT-decomposition requires use of the SVD. Whereas, calculating directly from the d-dimensional tensor only requires d simple tensor-vector multiplications.

Example 1 in 4-dimensions. Tables 11 and 12 display the numerical solutions to the integrations of equation 15 with decreasing step size's from the TT-decomposition and tensor respectively in 4-dimensions.

steps (step size)	Q	Absolute Error	time(s)
2 (0.5)	0.767883465177419	7.346289976880582e-004	0.000398
3 (0.3$\dot{3}$)	0.768619672550432	1.578375325217429e-006	0.000402
4 (0.25)	0.768618092382932	1.792174963988202e-009	0.000411
5 (0.2)	0.768618094176365	1.257993709202765e-012	0.000405

Table 11: Example 1, in 4 dimensions, from the TT-decomposition

The exact solution for $I(d)$ where d=4 is 0.768618094175107. Again for the smallest step size of 0.5 with three points we get a good approximation

steps (step size)	Q	Absolute Error	time(s)
2 (0.5)	0.767883465177420	7.346289976871701e-004	0.000401
3 (0.3$\dot{3}$)	0.768619672550432	1.578375325217429e-006	0.000399
4 (0.25)	0.768618092382932	1.792175075010505e-009	0.000412
5 (0.2)	0.768618094176365	1.258437798412615e-012	0.000410

Table 12: Example 1, in 4 dimensions, from the tensor

of $\times 10^{-4}$. When we reach the smallest step size of 0.2, we acquire a solution correct to 12 decimal places, which is the same as we acquired for the 3-dimensional case, although for 3-dimensions the error is slightly smaller, which is to be expected.

Comparing the results from the TT-decomposition to the tensor give almost identical results, although the tensor gives the best results at the largest step size, and unexpectedly the TT-decomposition gives the better result at the smallest step size. This is a different result to that achieved in the 3-dimensional case.

Also, it can be noted that the increase in dimension leads to the increase in the time to calculate the integrations. Again the TT-decomposition and the tensor give similar times over the decreasing step sizes.

Example 1 in 5-dimensions. Tables 13 and 14 display the numerical solutions to the integrations of equation 15 with decreasing step size's from the TT-decomposition and tensor respectively in 5-dimensions.

steps (step size)	Q	Absolute Error	time(s)
2 (0.5)	0.484485332277589	5.794491334570884e-004	0.000612
3 (0.3$\dot{3}$)	0.485066026526281	1.245115234249639e-006	0.000623
4 (0.25)	0.485064779997274	1.413772432456995e-009	0.000620
5 (0.2)	0.485064781412039	9.923173394099649e-013	0.000619

Table 13: Example 1, in 5 dimensions, from the TT-decomposition

steps (step size)	Q	Absolute Error	time(s)
2 (0.5)	0.484485332277589	5.794491334571994e-004	0.000631
3 (0.3$\dot{3}$)	0.485066026526280	1.245115234027594e-006	0.000622
4 (0.25)	0.485064779997274	1.413772821035053e-009	0.000618
5 (0.2)	0.485064781412039	9.925948951661212e-013	0.000627

Table 14: Example 1, in 5 dimensions, from the tensor

The exact solution for $I(d)$ where d=5 is 0.485064781411046. Again we get a really good result for the largest step size, calculated with 3 points, although gives a better approximation that with the same number of nodes for the 3 and 4-dimensional cases. Also if we look at the smallest step size, the result is again more accurate than that of that 3 and 4-dimensional cases.

Again comparing the TT-decomposition results to that of the tensor gives the result that the TT-decomposition gives the best approximation at every step size, which is unexpected due to the error caused via decomposing the tensor in the first place.

Example 1 in 6-dimensions. Tables 15 and 16 display the numerical solutions to the integrations of equation 15 with decreasing step size's from the TT-decomposition and tensor respectively in 6-dimensions.

steps (step size)	Q	Absolute Error	time(s)
2 (0.5)	0.109514751865740	1.571956327768298e-004	0.000835
3 (0.3$\dot{3}$)	0.109672285319574	3.378210568505402e-007	0.000841
4 (0.25)	0.109671947114936	3.835808615182401e-010	0.000837
5 (0.2)	0.109671947498786	2.689515277154442e-013	0.000829

Table 15: Example 1, in 6 dimensions, from the TT-decomposition

The exact solution for $I(d)$ where d=6 is 0.109671947498517. Clearly, increasing the dimensionality of the problem increases the accuracy as the 6-dimensional case gives the most accurate results over every step size.

steps (step size)	Q	Absolute Error	time(s)
2 (0.5)	0.109514751865740	1.571956327770380e-004	0.000837
3 (0.3$\dot{3}$)	0.109672285319574	3.378210566840068e-007	0.000829
4 (0.25)	0.109671947114936	3.835809447849670e-010	0.000832
5 (0.2)	0.109671947498786	2.690347944422911e-013	0.000840

Table 16: Example 1, in 6 dimensions, from the tensor

Also, increasing the dimensionality leads to the TT-decomposition giving a more accurate result than that of the tensor. This is the opposite of what we would expect due to the increasing dimensionality leads to an increased error in the TT-decomposition.

The time taken to compute the calculations increases as the dimensions increase, although the step size has no effect on the efficiency of the calculations.

3.1.2 Example 2

For This example I will consider the d-dimensional standard normal distribution, which in one-dimension is described by the probability density function:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{16}$$

Where parameters μ and σ^2 are the mean and the variance [14]. The distribution with $\mu = 0$ and $\sigma^2 = 1$ is called standard normal and these are the parameters I will be using. Hence we will acquire the following equation:

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \tag{17}$$

We know that for the one dimensional case, 68% of the data will be within 1 standard deviation, $\sigma = 1$, of the mean. 95% of the data will fall within 2 standard deviations, $2\sigma = 2$, of the mean and almost 97% of the data will fall within 3 standard deviations, $3\sigma = 3$.

This can of course be extended to a multiple dimensional case, which is known as the multivariate normal distribution or multivariate Gaussian distribution.

If we firstly consider the 2-dimensional, bivariate case. The probability density function is given by:

$$f(x,y) = \frac{1}{2\pi\sigma_x\sigma_y\sqrt{1-\rho^2}} e^{-\frac{1}{2(1-\rho^2)}\left[\frac{(x-\mu_x)^2}{\sigma_x^2}+\frac{(y-\mu_y)^2}{\sigma_y^2}-\frac{2\rho(x-\mu_x)(y-\mu_y)}{\sigma_x\sigma_y}\right]} \tag{18}$$

Where $\rho = \text{covarience}(\sigma_x, \sigma_y)/\sigma_x\sigma_y$, is the correlation between x and y [14]. Hence if we apply the same standard parameters we get:

$$f(x,y) = \frac{1}{2\pi\sqrt{1-\rho^2}} e^{-\frac{x^2-2\rho xy+y^2}{2(1-\rho^2)}} \tag{19}$$

The general multivariate case for d-dimensions has the equation:

$$f(X,\mu,\Sigma) = \frac{1}{\sqrt{|\Sigma|\,(2\pi)^d}} e^{-\frac{(x-\mu)'\Sigma^{-1}(x-\mu)}{2}} \tag{20}$$

Where X and μ are vectors of length d and Σ is a positive definite $d \times d$ matrix [14]. If we apply our standard parameters, and so letting Σ equal a $d \times d$ identity matrix, I_d, this becomes :

$$f(x_1, x_2, ..., x_d) = \frac{1}{\sqrt{(2\pi)^d}} e^{-\frac{X'X}{2}} \tag{21}$$

In equation 21 $X = [x_1 x_2 ... x_d]'$.

I will now apply the trapezium rule to this problem:

$$\int_a^b ... \int_a^b f(x_1, ..., x_d) dx_1 ... dx_d$$

$$\approx \frac{h}{2} \sum_{k_1=1}^{n} ... \sum_{k_d=1}^{n} w_{k_1} ... w_{k_d} f\left((x_{k_1}, ..., x_{k_d})\right)$$

$$\text{where} \quad w_{k_i} = 2 \ \text{ for } \ i \neq 1, n, \quad w_{k_i} = 1 \ \text{ for } \ i = 1, n.$$

Example 2, between -1 and 1 ($-\sigma$ to σ). I will firstly be analysing the standard multivariate normal distribution between -1 and 1.

In 2 dimensions (variables), between -1 and 1 ($-\sigma$ to σ). This is the standard bivariate case and so we know that the exact solution we are looking for is going to be 68% of 68%, which is 47%, or more precisely 46.60649%.

steps (step size)	Q	Absolute Error
3 (1)	0.410769479876322	0.055295462123678
4 ($0.6\dot{6}$)	0.441548891468991	0.024516050531009
5 (0.5)	0.452285610783585	0.013779331216415
6 (0.4)	0.457249316967022	0.008815625032978
7 ($0.3\dot{3}$)	0.459944149847137	0.006120792152863
8 (0.2857)	0.461568547371707	0.004496394628293
9 (0.25)	0.462622643958043	0.003442298041957
10 ($0.2\dot{2}$)	0.463345238793662	0.002719703206338
11 (0.2)	0.463862061760540	0.002202880239460
12 ($0.\dot{1}\dot{8}$)	0.464244428297124	0.001820513702876
13 ($0.1\dot{6}$)	0.464535235765965	0.001529706234035
14 (0.1538)	0.464761544217568	0.001303397782432
15 (0.1429)	0.464941107983212	0.001123834016788

Table 17: Example 2, in 2 dimensions from the tensor

As you can see from tables 17 and 18, the TT-decomposition and the tensor contract to give exactly identical solutions at every step size. This is due to the fact that the standard multivariate normal distribution is a rank one tensor and so decomposes with zero error into a tensor train. This is caused by having zero correlation between the variables, which means they are independent, leading to a perfect TT-decomposition. I will now alter the parameters μ and Σ so that we should get a tensor that isn't of rank 1.

steps (step size)	Q	Absolute Error
3 (1)	0.410769479876322	0.055295462123678
4 ($0.6\dot{6}$)	0.441548891468991	0.024516050531009
5 (0.5)	0.452285610783585	0.013779331216415
6 (0.4)	0.457249316967022	0.008815625032978
7 ($0.3\dot{3}$)	0.459944149847137	0.006120792152863
8 (0.2857)	0.461568547371707	0.004496394628293
9 (0.25)	0.462622643958043	0.003442298041957
10 ($0.2\dot{2}$)	0.463345238793662	0.002719703206338
11 (0.2)	0.463862061760540	0.002202880239460
12 ($0.\dot{1}\dot{8}$)	0.464244428297124	0.001820513702876
13 ($0.1\dot{6}$)	0.464535235765965	0.001529706234035
14 (0.1538)	0.464761544217568	0.001303397782432
15 (0.1429)	0.464941107983212	0.001123834016788

Table 18: Example 2, in 2 dimensions from the TT-decomposition

Example 2, with non-standard parameters, between $\mu_i - \sigma_i$ and $\mu_i + \sigma_i$ in the ith dimension. As we are altering the correlations, means and standard deviations we will acquire a slightly different probability. If you consider figures 7 and 8 which have correlations 0 and 0.9 respectively. Each figure has the probability density function displayed from both the $x_1 = x_2$ angle and the $x_2 = x_1$ angle, which have means $\mu_1 = 1, \mu_2 = 0$ and standard deviations $\sigma_1^2 = 1$ and $\sigma_2^2 = 4$.

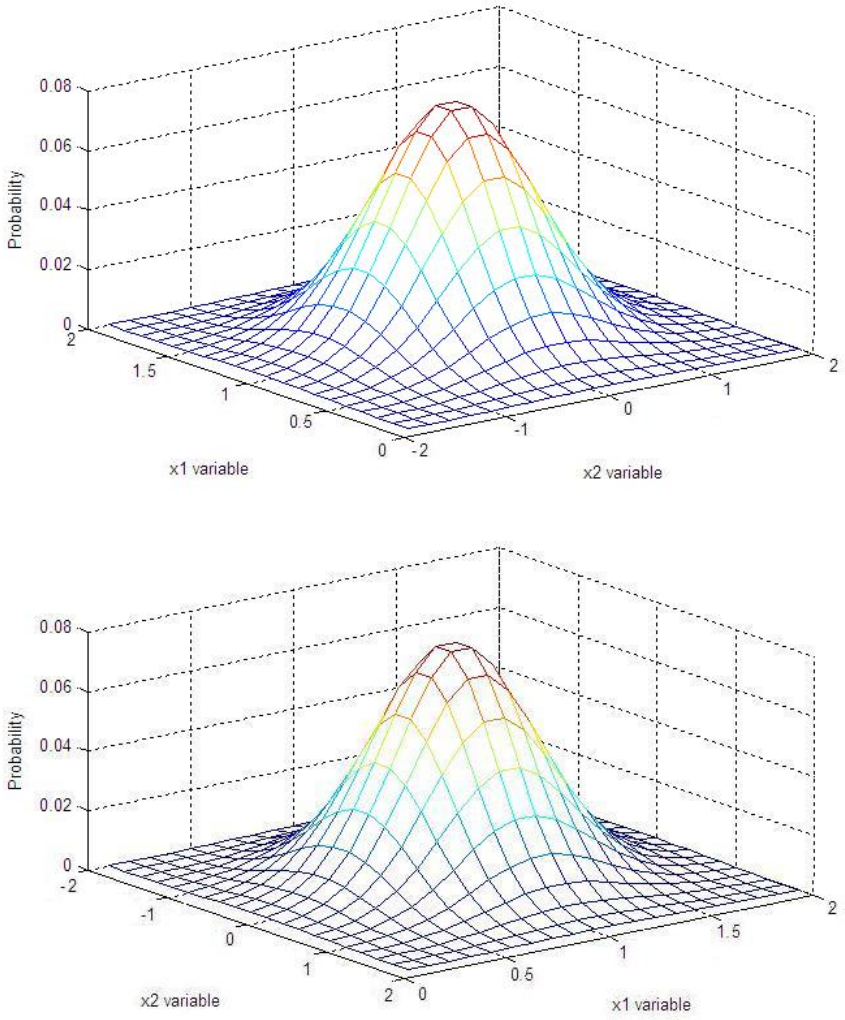


Figure 7: bivariate with zero correlation

The increase in correlation leads to the different shape of the probability density function, you can clearly see that the increase in correlation leads to a larger probability at the center μ_1, μ_2, this comes form the fact that an increase in correlation begins to constrict the probability density function about the $x_1 = x_2$ plane. As we are calculating our integral about $\mu \pm \sigma$ we should acquire a larger probability than we previously did. Ofcourse if

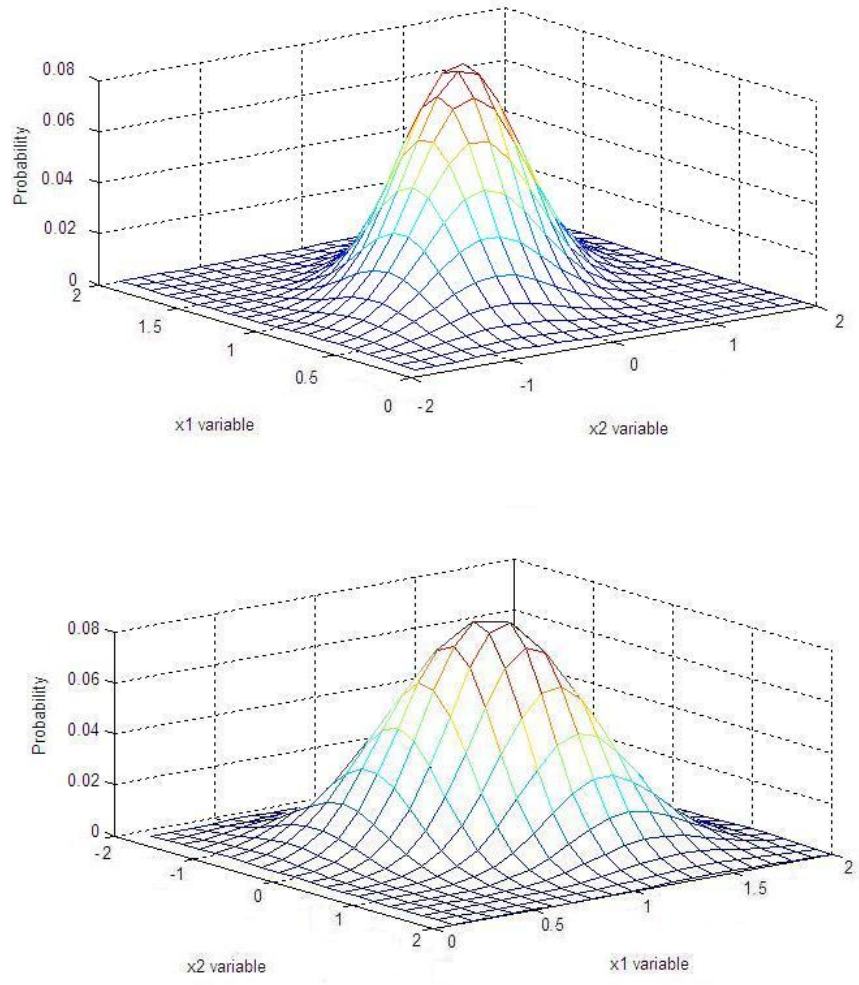


Figure 8: bivariate with correlation 0.9

we to integrate from $-\infty$ to ∞ we would get 1 regardless of the starting parameters.

Example 2, in 2-dimensions. Due to the use of a non set of non-standard parameters, a different μ and Σ, the distribution will shift accordingly to center at μ with a variation of σ. For the 2-dimensional case I have let

$$\mu = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \qquad \Sigma = \begin{bmatrix} 1 & 0.3 \\ 0.3 & 2 \end{bmatrix}.$$

Hence, due to the correlation, we will achieve a slightly different percentage of 47.14143%, to that which would be achieved by the standard normal distribution. Also a change of the integration limits is required, we must now integrate form -1 to 1 in the x_1 dimension and from $1-\sqrt{2}$ to $1+\sqrt{2}$ in the x_2 dimension. These are obtained from $\mu \pm \sigma$.

In this example I have included the time taken the time taken to complete the TT-decomposition along with the contraction time, as we saw that the difference in contraction times was negligible.

steps	Q	Absolute Error	time(s)
3	0.414393921240724	0.057020401753395	0.000437
4	0.446112899732317	0.025301423261802	0.000516
5	0.457190030235556	0.014224292758563	0.000560
6	0.462313068201437	0.009101254792682	0.000639
7	0.465094899106634	0.006319423887485	0.000728
8	0.466771902389441	0.004642420604678	0.000832
9	0.467860201204107	0.003554121790012	0.000945
10	0.468606271616959	0.002808051377160	0.001082
11	0.469139899747050	0.002274423247069	0.001219
12	0.469534707273079	0.001879615721040	0.001371
13	0.469834981148493	0.001579341845626	0.001538
14	0.470068659084757	0.001345663909362	0.001736
15	0.470254071799575	0.001160251194544	0.001902

Table 19: Example 2, in 2 dimensions from the tensor

steps	Q	Absolute Error	time(s)
3	0.414393921240724	0.057020401753395	0.000657
4	0.446112899732317	0.025301423261802	0.000712
5	0.457190027503422	0.014224295490697	0.000795
6	0.462313062884608	0.009101260109511	0.000867
7	0.465094892703025	0.006319430291094	0.000963
8	0.466771895800884	0.004642427193235	0.001070
9	0.467860194864586	0.003554128129533	0.001198
10	0.468606265707991	0.002808057286128	0.001333
11	0.469139894325028	0.002274428669091	0.001488
12	0.469534702335121	0.001879620658998	0.001651
13	0.469834976665403	0.001579346328716	0.001851
14	0.470068655017327	0.001345667976792	0.002044
15	0.470254068106520	0.001160254887599	0.002255

Table 20: Example 2, in 2 dimensions from the TT with error 0.0o1

steps	Q	Absolute Error	time(s)
3	0.414375191391604	0.057039131602515	0.000643
4	0.446094948117974	0.025319374876145	0.000705
5	0.457180952375195	0.014233370618924	0.000779
6	0.462309112155652	0.009105210838467	0.000862
7	0.465093477034532	0.006320845959587	0.000957
8	0.466771576663234	0.004642746330885	0.001066
9	0.467860190454007	0.003554132540112	0.001191
10	0.468606150829030	0.002808172165089	0.001326
11	0.469139439638143	0.002274883355976	0.001472
12	0.469533788351477	0.001880534642642	0.001649
13	0.469833546648905	0.001580776345214	0.001831
14	0.470066688488310	0.001347634505809	0.002014
15	0.470251565562013	0.001162757432106	0.002228

Table 21: Example 2, in 2 dimensions from the TT with error 0.01

It is clearly visible from tables 19, 20 and 21 that calculating the contraction directly from the tensor is far more efficient than that of the TT-decomposition. This is due to need to decompose the tensor to begin with. Another observation that can be made from the times taken to compute these contractions is that decreasing the step size leads to an increase in the computation time. This wasn't the case with example 1, where there were all very similar. This may be due to using a different numerical scheme, with different quadrature weights. Also, the errors are much larger for this example when comparing them to that of example 1. This may again be due to the choice of quadrature.

The tensor gives the smallest errors of the three tables, which is as we expect, because there has been no error introduced by decomposing the tensor into a tensor train. Also, when using the less accurate TT-decomposition in table 21, there is again a further loss of accuracy. This is caused by using a less accurate tensor train, and so there is already a larger error before we begin to compute the contraction. This loss of accuracy is compensated by an increase in the efficiency.

Example 2, in 3-dimensions. For the 3-dimensional case I have let

$$\mu = \begin{bmatrix} 0 \\ 1 \\ -0.5 \end{bmatrix} \qquad \text{and} \qquad \Sigma = \begin{bmatrix} 1 & 0.3 & 0.1 \\ 0.3 & 2 & -0.4 \\ 0.1 & -0.4 & 1.5 \end{bmatrix}.$$

Again, due to the correlation's, we will achieve a different percentage, this time of 82.0579%.

In tables 22, 23 and 24, you can again see that the 3-dimensional case, similarly to the 2-dimensional case that calculating the contraction directly form the tensor is more efficient, although this time it is much more efficient, this is due to the rime taken to decompose a tensor increases exponentially as the number of dimensions increases. As expected the 2-dimensional case is quicker than that of the 3-dimensional case, as we saw in example 1.

Again the tensor gives the smallest errors of the three tables. When comparing to the 2-dimensional case, as the step size decreases, the contraction of the tensor trains give a larger difference in error, again caused by the error introduced by decomposing the tensor, which increases with increased dimensionality. Also, the less accurate TT-decomposition in table 24, leads to a further loss of accuracy. Again caused by using a less accurate tensor train.

steps	Q	Absolute Error	time(s)
3	0.674409032016107	0.146169597802947	0.000660
4	0.754510492246537	0.066068137572517	0.000930
5	0.783211099959438	0.037367529859616	0.001635
6	0.796609576614736	0.023969053204319	0.002014
7	0.803917825445111	0.016660804373943	0.002930
8	0.808334660576165	0.012243969242889	0.004176
9	0.811205451161317	0.009373178657737	0.005840
10	0.813175515797560	0.007403114021495	0.007630
11	0.814585618897129	0.005993010921926	0.010263
12	0.815629433297236	0.004949196521818	0.012932
13	0.816423622967231	0.004155006851823	0.016239
14	0.817041858540985	0.003536771278069	0.020306
15	0.817532514931975	0.003046114887080	0.024855

Table 22: Example 2, in 3 dimensions from the tensor

steps	Q	Absolute Error	time(s)
3	0.674409032016107	0.146169597802947	0.001045
4	0.754510492246537	0.066068137572517	0.001357
5	0.783211090055731	0.037367539763323	0.001882
6	0.796609550161228	0.023969079657826	0.002541
7	0.803917792104603	0.016660837714451	0.003517
8	0.808334626503367	0.012244003315688	0.004937
9	0.811205419255614	0.009373210563440	0.006660
10	0.813175487182350	0.007403142636704	0.009994
11	0.814585593832091	0.005993035986964	0.011186
12	0.815629411649693	0.004949218169361	0.014501
13	0.816423604444332	0.004155025374722	0.018131
14	0.817041842801271	0.003536787017783	0.022677
15	0.817532501637163	0.003046128181891	0.027940

Table 23: Example 2, in 3 dimensions from the TT with error 0.001

steps	Q	Absolute Error	time(s)
3	0.674409032016107	0.146169597802947	0.001047
4	0.754501937645082	0.066076692173972	0.001338
5	0.783212872384791	0.037365757434263	0.001842
6	0.796618155884882	0.023960473934173	0.002542
7	0.803926341365658	0.016652288453397	0.003624
8	0.808339556192410	0.012239073626644	0.004785
9	0.811205113842517	0.009373515976537	0.006523
10	0.813169351140584	0.007409278678470	0.008751
11	0.814573549435406	0.006005080383648	0.011809
12	0.815611636493951	0.004966993325103	0.014554
13	0.816400394282805	0.004178235536249	0.018009
14	0.817013539619649	0.003565090199405	0.023063
15	0.817499456127917	0.003079173691137	0.027662

Table 24: Example 2, in 3 dimensions from the TT with error 0.01

4 Conclusions and Further Work

The TT-decomposition formula will decompose a tensor with zero error if its auxiliary unfolding matrices are of low rank. This is always the case when a tensor has a low canonical rank. Otherwise this formula can represent a tensor with minimal accuracy.

The algorithm that we used for the TT-decomposition has room for improvement, although we obtained accurate results fairly efficiently the use of the truncated SVD for large scale dense unfolding matrices is clearly unfordable in higher dimensions. There are several options that can be considered in replacing the SVD, such as a dyadic decomposition or the QR decomposition [12]. This is an area in which further work could be done to construct an improved algorithm and compare times taken and accuracy to that of the currently used TT-decomposition algorithm.

When comparing the times taken between the tensor and the tensor train to calculate the high-dimensional in integration there were only marginal differences. This is due to the time taken to calculate the tensor train being included, therefore anytime that is saved within the quadrature is used in the decomposition stage. Although reducing the time taken is not the goal of using the tensor train. The reason for using the tensor train is to dramatically reduce the amount of computational memory required.

The high-dimensional integration of a tensor with a known low rank, will give more accurate results as the dimensionality of the tensor increases. With increasing dimensionality d the tensor train gives more accurate results than that of the tensor itself. This is due to the low rank of the tensor, which leads to a perfect decomposition into the tensor train.

When we compute the high-dimensional integration of a tensor with a higher rank, we acquire the opposite result. The increase in dimensionality leads to the increase in error of the tensor train, and increased computation time. This is caused by the initial error introduced by computing the TT-decomposition.

5 Appendix

5.1 M-files

5.1.1 tensor matrix multiplication

```
function[Output]= tensorxmatrix(A,M,n)
% Tensor A, Matrix M, n is mode of multiplication
% flatten a into n-mode matrix
B= shiftdim(A,n-1);
C= B(:,:);
% create target dimensions of the output
outdim = size(B);
outdim(1) = size(M,1);
% Multiuply the matrices
Output = M*C;
% unflattened into correct tensor shape
Output = reshape(Output,outdim);
% change dimensions back to get correct output
Output = shiftdim(Output,length(outdim)-n+1);
```

5.1.2 tensor vector multiplication

```
function[Output]= tensorxvector(A,V,n)
% Tensor A, Vector V, n is mode of multiplication
%flatten a into n-mode matrix
B= shiftdim(A,n-1);
C= B(:,:);
% create target dimensions of the output
outdim = size(B);
outdim(1) = 1;
% Multiuply the matrices
Output = V'*C;
% unflattened into correct tensor shape
Output = reshape(Output,outdim);
% change dimensions back to get correct output
Output = shiftdim(Output,length(outdim)-n+1);
```

5.1.3 truncated SVD

```
function[U,S,V,ranknew]=truncsvd(M,errorbound)
% calculate the truncated SVD of matrix M
% to satisfy the errorbound
ranknew=rank(M);
[U,S,V]=svd(M);
for i = 1 : min(size(M))
sum=0;
for j =i+1 : min(size(M))
sum=sum+S(j,j)^2;
b = sqrt(sum);
end
if le(b,errorbound)==1
ranknew = i;
break
else
end
end
```

5.1.4 tensor inner product

```
function[Output] = tensor_inner_product(A,B)
% compute the inner product of Tensors A and B
Output=A(:)'*B(:);
```

5.1.5 tensor outer product

```
function[Output] = tensor_outer_product(A,B)
% compute the inner product of Tensors A and B
c=A(:)*B(:)';
% The A(:) and B(:) vectorize A and B, then c
% is the outer product of two vectors
Output=reshape(c,[size(A) size(B)]);
% c is then reshaped to the correct dimensions,
% which are calculated by
% size(A) and size(B)
```

5.1.6 Sparse tensor generator

```
function[A]=sparsetensor(A,d)
n = d;
elements=3^n;
for j = 1 : ceil(elements.*rand(1,1))
%ceil(n.*rand(1,1)) must be added for each dimension
%so that a random element is generated in a random position
A(ceil(n.*rand(1,1)),ceil(n.*rand(1,1)),ceil(n.*rand(1,1))) = rand;
end
```

5.1.7 tensor contracted product

```
function[Output] = tensor_contracted_product(A,B,da,db)
% contracts tensors A and B in their
% da, and db dimensions respectively,
dimA=ndims(A);
dimB=ndims(B);
% Generate the size of the output tensor
for k=1:da-1
sizeout1(1,k)=size(A,k);
end
for k=da+1:dimA
sizeout1(1,k-1)=size(A,k);
end
for k=1:db-1
sizeout2(1,k)=size(B,k);
end
for k=db+1:dimB
sizeout2(1,k-1)=size(B,k);
end
% reshape the tensors in their da and db modes
P= shiftdim(A,da-1);
C= P(:,:);
D=shiftdim(B,db-1);
E= D(:,:);
% Perform the multiplication and reshape Output=C'*E;
Output=reshape(Output,[sizeout1 sizeout2]);
```

5.1.8 tensor train decomposition

```
function[]=ttdecomp(A,error)
%dimensions of the tensor.
dimA=ndims(A);
%create dimA carriage's using the cell function.
tt=cell(dimA,1);
%calculate the frobenious norm.
nrm=sqrt(A(:)'*A(:));
%create a vector r, to store the ranks of each step.
r=zeros(1,dimA-1);
%calculate the error bound using the given formula
errorbound=((error*nrm)/sqrt(dimA-1));
%calculate the dimensions of the first carriage.
Nl=size(A,1);
Nr=numel(A)/Nl;
%create a dummy tensor B.
B=A;
%reshape B to perform truncated svd.
M=reshape(B,[Nl,Nr]);
%perform the truncated svd to decompose the matrix, and
%to aquire its approximate rank which satisfies the
%error bound.
[U,S,V,ranknew]=truncsvd(M,errorbound);
r(1,1)=ranknew;
%here we 'crop' the matrices as to only use ranknew
%orthonormal columns of U, S and V.
U=U(:,1:ranknew);
S=S(1:ranknew,1:ranknew);
%recalculate M for next carriage.
M=V(:,1:ranknew)*S;
M=M';
%store the first carriage in the first cell.
tt1=U;
```

```
  %calculate the remaining carriage's.
for k=2:dimA-1
%calculate the new dimensions
Nl=size(A,k);
Nr=Nr/Nl;
%reshape acording to new dimensions and rank.
M=reshape(M,[r(1,k-1)*Nl,Nr]);
%perform the truncated svd to decompose the matrix,
%and to aquire its
%approximate rank which satisfies the error bound.
[U,S,V,ranknew]=truncsvd(M,errorbound);
r(1,k)=ranknew;
%agian we 'crop' the matrices as to only use ranknew
%orthonormal columns of U, S and V.
U=U(:,1:ranknew);
S=S(1:ranknew,1:ranknew);
M=V(:,1:ranknew)*S;
M=M';
%reshape U to its correct size and then store the kth
%carriage, in the kth cell.
U=reshape(U,[r(1,k-1),Nl,r(1,k)]);
ttk=U;
end
%the final carriage is simply the final M,
% and so we store it.
ttdimA=M';
%this final section simply prints the output of the
%tensor train carriage sizes, and displays them aswell.
tt
for q=1:dimA
ttq
end
end
```

5.1.9 Multiplying out a tensor train

```
function[T] = ttmult(tt)
dimtt=size(tt,1);
T=tensorxmatrix(tt2,tt1,1);
for j=2:dimtt-1
if j==dimtt-1
T=tensorxmatrix(T,ttdimtt,j+1);
break
else
t=ndims(T);
T=tensor_contracted_product(T,ttj+1,t,1);
end
end
```

5.1.10 TT Contraction (Integration)

```
function[]=ttcontraction(steps,decomperror)
q=steps;
% Dependant on the type of quadrate chosen the weights and
% Abscissas will need to be entered as w and Z in the form
% of a column vector, of length q.
Z=[ z1;
...;
zq];
w=[w1;
...;
wq];
% the fnction must the be calculated with d for loops to
% create the d-dimensional tensor.
for x1=1:q
...
for xd=1:q
A(x1,..,xd)=f(x1,...,xd) end
...
end
% set B to be a dummy tensor and then recalculate the
```

```
% tensor dependant on the Abscissas.
B=A;
d=ndims(A);
n=size(A,1);
for x1 = 1: n,
...
for xd=1:n
B(x1,...,xd)=f(Z(x1),...,Z(xd))
end,
...
end
% calculate the tt decomposition
tt=ttdecomp(B,decomperror);
% apply the chosen type of quadrature to tensor train.
v=tensorxvector(tt1,w,1)';
for k= 2 : d
W = tensorxvector(ttk,v,1);
W = reshape(W,[size(ttk,2),size(ttk,3)] );
v = (W'*w);
end
```

References

[1] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions, With Formulas, Graphs and Mathematical Tables*. Bernan Assoc, 1972.

[2] I. Avramidi. Vector analysis. *Lecture Notes, New Mexico Institute of Mining and Technology*, 2008.

[3] B. W. Bader and T. G. Kolda. Algorithm 862: Matlab tensor classes for fast algorithm prototyping. *ACM Trans*, Vol. 32, No. 4 , pp. 635-653, 2006.

[4] T.S. Blyth and E.F. Robertson. *Basic Linear Algebra*. Springer, second edition edition, 2005.

[5] N. V. Boulgouris, K. N. Plataniotis, and E. Micheli-Tzanakou. *Biometrics: Theory, Methods, and Applications*. Wiley-IEEE,, volume 9 of ieee press series on computational intelligence edition, 2009.

[6] M. J. Crowe. *A History of Vector Analysis*. University of Notre Dame Press, 1967.

[7] H. F. Davis and A. D. Snider. *Introduction to Vector Analysis*. Wm. C. Brown Publishers, sixth edition edition, 1991.

[8] T. G. Kolda and B. W. Bader. Tutorial on matlab for tensors and the tucker decomposition, 2005.

[9] T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM*, 2008.

[10] J. B. Kruskal. *Rank, decomposition, and uniqueness for 3-way and N-way arrays*. North-Holland, Amsterdam, 1989.

[11] I. V. Oseledets. Compact matrix form of the d-dimensional tensor decomposition. *INM RAS*, 2009-01, 2009.

[12] I. V. Oseledets and E. Tyrtyshnikov. Linear algebra and its applications. *ELSEVIER*, 432 (2010) 7088, 2009.

[13] M. R. Spiegel. *Theory and Problems of Vector Analysis*. Schaum's Outline Series. McGraw-Hill Book Co., 1974.

[14] Y. L. Tong. *The multivariate normal distribution*. Springer series in statistics. Springer-Verlag, 1990.